



Proyecto Fin de Carrera

Control digital con FPGA de actuador lineal en lazo cerrado

Autor

José Luis Ruiz Gómez

Director

José Ignacio Artigas

Convocatoria

Junio 2013

Titulación

Ingeniería Técnica Industrial

Especialidad Electrónica Industrial

RESUMEN

Este proyecto sigue con el desarrollo del control de un prototipo de actuador lineal para utilizarlo como emulador de equinoterapia, que comenzó otro alumno como proyecto fin de carrera.

Hasta el momento éste constaba de un motor de continua alimentado por un puente en H mediante la técnica PWM y controlado por una FPGA, que le mandaba la señal de PWM y el sentido de giro. Ahora, el propósito es continuar con la mejora del proyecto, aplicando un control en bucle cerrado que seguirá una trayectoria recibida mediante el envío en serie de datos a través del ordenador.

Además también se ha mejorado la placa de potencia, haciéndola más compacta y estrecha, debido a que, aunque sólo se controle un motor, el objetivo final es el movimiento de una silla de caballo con 6 motores. Por tanto, se configura pensando en su posible ampliación.

ÍNDICE

CAPITULO 1 5

1 Introducción 5

1.1 Objetivos 5

1.2 Motivación del proyecto. 11

1.3 Diagrama de bloques del proyecto. 12

1.4 Estructura del presente documento. 13

CAPÍTULO 2 14

2 Control en bucle cerrado 14

2.1 Motor de corriente continua 14

2.2 Sistema físico. 14

2.3 Bucle cerrado 17

CAPÍTULO 3 21

3 Diseño digital en VHDL. 21

3.1 Introducción. 21

3.2 UART emisor. 22

3.3 UART receptor. 24

3.4 Habilitación del reloj del emisor y receptor. 27

3.5 Memoria RAM. 28

3.6 Máquina de estados para el control del motor según las instrucciones recibidas. 30

3.7 Máquina de estados del motor según si es bucle abierto o bucle cerrado. 32

3.8 Preparación del dato de referencia desde la memoria RAM hasta el bucle cerrado. 33

CAPÍTULO 4 41

4 Programación en MATLAB para el envío y recepción de datos desde el ordenador a la FPGA. 41

4.1 Senoide. 43

4.2 Escalón. 48

CAPÍTULO 5 49

5 Conexión del actuador. 49

5.1 FPGA utilizada. 49

5.2 Interfaz de usuario de la FPGA. 49

5.3 Conexión del encoder. 51

5.4 Finales de carreras FCI y FCS. 51

CAPÍTULO 6 52

6 Simulaciones 52

CAPÍTULO 7 55

7 PCB 55

7.1 Reducción de tamaño 55

7.2 Mejoras 58

CAPÍTULO 8 60

8 Anexos 60

8.1 Planos 60

8.2 Manual de instrucciones 62

8.3 Código VHDL 65

8.3.1 TOP 65

8.3.2 MOTOR_DC 85

8.3.3 UART_TX 106

8.3.4 UART_RX 109

8.4 Programación MATLAB 114

8.4.1 SENOIDE 114

8.4.2 ESCALÓN 118

8.5 Bibliografía 123

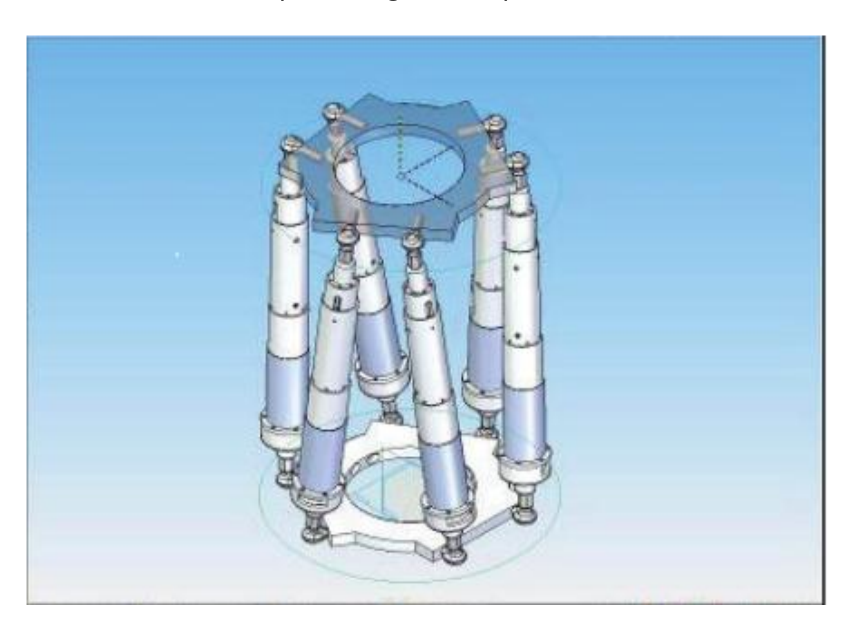
8.6 Programas utilizados. 123

CAPITULO 1

1 Introducción

1.1 Objetivos

El objetivo es avanzar en el desarrollo de un emulador de equinoterapia, que estará compuesto por varios actuadores lineales controlados de forma sincronizada desde un PC de forma que el asiento del emulador pueda seguir la trayectoria deseada.



Emulador, diseño realizado en Solid Edge de la plataforma completa.

En este proyecto se trabaja con uno de los actuadores que está formado por un motor de corriente continua, una FPGA para el control del mismo, una placa de potencia y una fuente de alimentación. El punto de partida es el proyecto final de carrera de Javier Marco Estruc llamado CONTROL CON FPGA DE ACTUADOR LINEAL PARA EMULADOR DE EQUINOTERAPIA, en el cual, con el diseño digital en VHDL en una FPGA se manejaba el actuador lineal mediante un puente en H en bucle abierto.

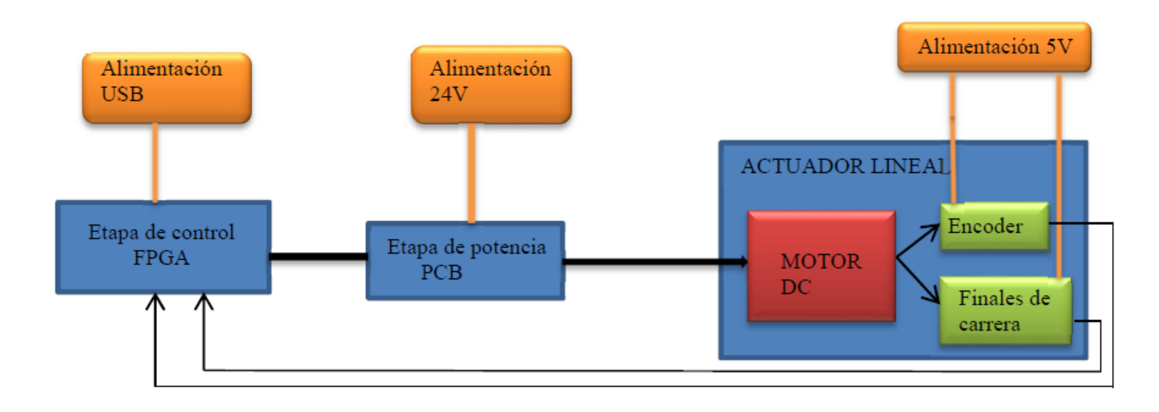


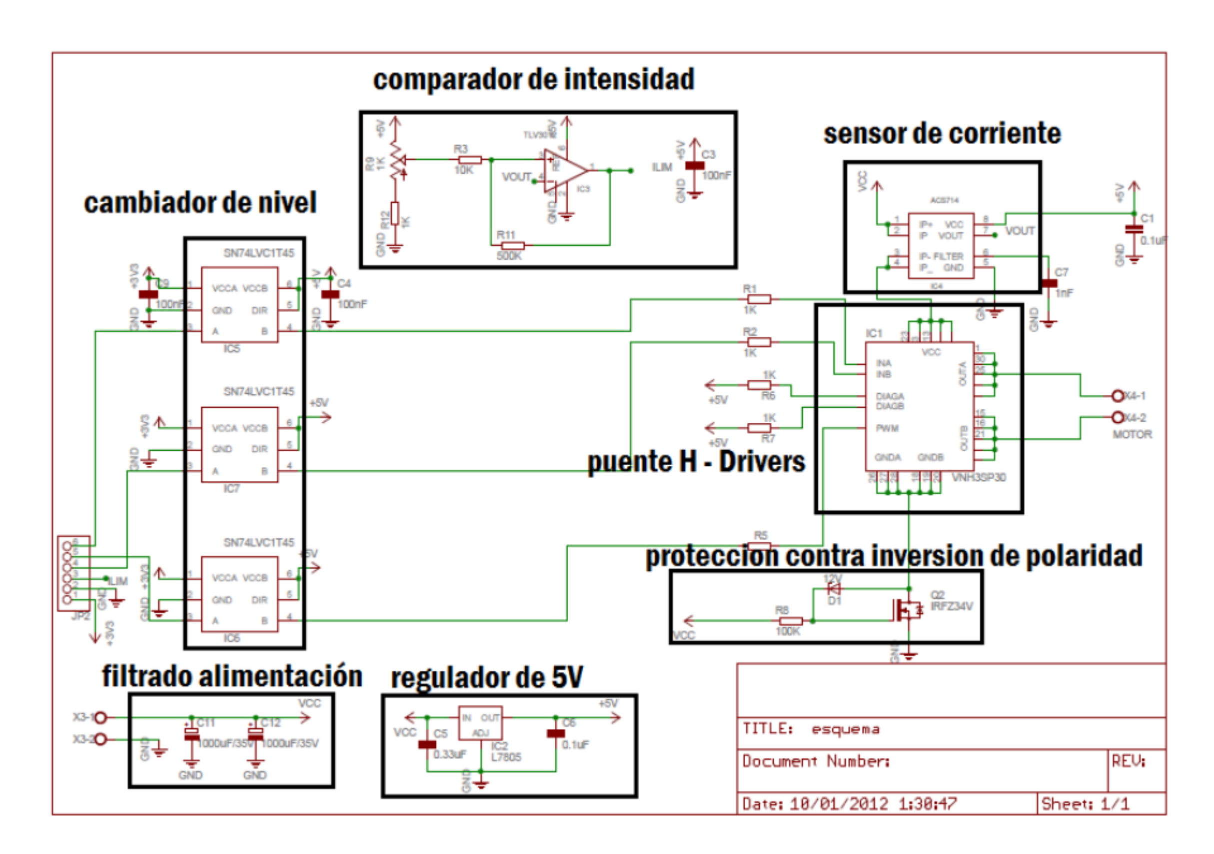
Diagrama de bloques general del proyecto de Javier Marco.



Proyecto final de carrera de Javier Marco. En la derecha se encuentra el actuador lineal, la placa verde es la FPGA, la placa marrón es la placa de potencia y la fuente de alimentación es lo de la izquierda.

El actuador lineal tiene un rango de 10 cm. de carrera, a ambos extremos se encuentran dos finales de carrera, el superior y el inferior. Solidario al eje del motor se sitúa un encoder que permite conocer la posición de éste. Tanto el encoder, como los finales de carrera se conectan a la FPGA.

Para el control del actuador lineal se necesita dos circuitos, el de control, diseñado en VHDL en la FPGA y el de potencia. La técnica para el control de la velocidad del motor es la de PWM que se explica posteriormente.



Esquema de la placa de potencia de Javier Marco.

La placa de potencia consta de:

- Un puente en H que permite cambiar el giro del motor encapsulado en el circuito SMD VNH3SP30-E
- Una protección contra la inversión de polaridad para que en el caso de inversión de polaridad de la fuente no se rompa el integrado VNH3SP30-E
- Un sensor de corriente que junto con el comparador de intensidad permite actuar frente a una sobrecorriente
- Un cambiador de nivel que permite adaptar el estándar de 3V3 con que trabaja la FPGA a 5V con la que trabaja el puente en H.
- Una fuente de alimentación de 5V para alimentar todos los integrados que funcionan con cinco voltios.

Con la FPGA se controla el giro del motor con las dos señales (INA para que gire hacia arriba e INB para que gire hacia abajo) y la velocidad del motor mediante la técnica PWM que se refleja en la salida PWM.

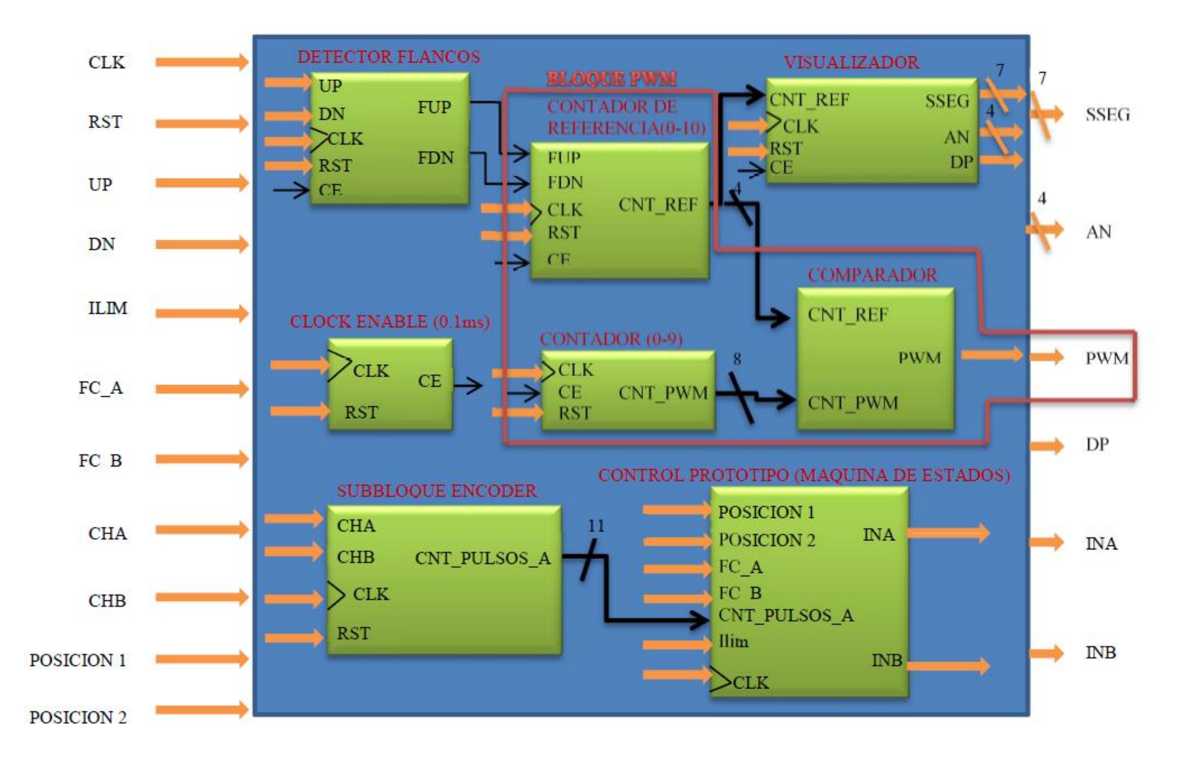
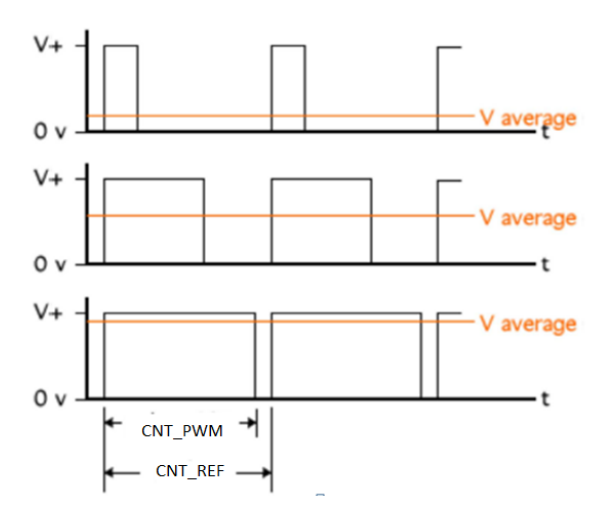


Diagrama de bloques del diseño en VHDL de Javier Marco.

El diseño en VHDL consta de:

- Bloque PWM, éste se implementa mediante dos contadores, uno de referencia que cuenta de 0 a 10 con 10 kHz de frecuencia y otro que fija el tiempo en ON de la señal PWM de 0 a 9. Ambos se comparan para generar la señal PWM.



- Detector de flancos, mediante dos pulsadores UP y DN se fija en el contador CNT_PWM el tiempo en ON de la señal en PWM.

- Subbloque encoder, que permite conocer el número de pulsos recibidos desde el encoder.

- Máquina de estados para el control del sentido de giro del motor y que tiene implementado 4 movimientos, uno de subida, otro de bajada, otro de subida o bajada en escalón de 10mm. y otro movimiento que repite el ciclo de subida y bajada de 20 mm.

El interfaz de usuario se realiza mediante los pulsadores y switches de la FPGA.



SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0		BTN3	BTN2	BTN1	BTN0
ILIM Disable	FC Disable			CICLO	STEP	BAJA	SUBE		RST		DN	UP

Las señales más importantes para la placa de potencia son INA, INB y PWM que se generan en la FPGA. INA indica que el sentido de giro para el motor tiene que ser hacia arriba, INB indica que el sentido de giro tiene que ser hacia abajo, y PWM es la técnica de control utilizada en este diseño que marca la velocidad con la que gira el motor.

Los pulsadores UP y DN aumentan o disminuyen el contador CNT_PWM que almacena el tiempo en ON de la señal PWM y se refleja en el visualizador de 7 segmentos. Cuanto más alto sea más velocidad tendrá actuador lineal. Para 0 la velocidad es nula, con 9 se consigue la velocidad más alta.

El SW0 y SW1 fija el sentido de bajada y de subida respectivamente.

Para SW2, primero se indica la velocidad con DN y UP, después con el switch activado si se vuelve a pulsar UP, el motor subirá 10 mm y si se pula DN, el motor bajará 10 cm.

Con la activación del SW3 se consigue un movimiento repetitivo de subida y bajada de 20 mm.

Hasta aquí se ha explicado el proyecto de partida.

Las mejoras a realizar para el nuevo proyecto son principalmente 3:

- Control en bucle cerrado del motor, partiendo del control en bucle abierto ya realizado.
- Comunicación en serie de la FPGA con el ordenador, de donde recibirá las ordenes.
- Compactar la placa de potencia para que se pueda conectar más de una placa a la misma placa de FPGA.

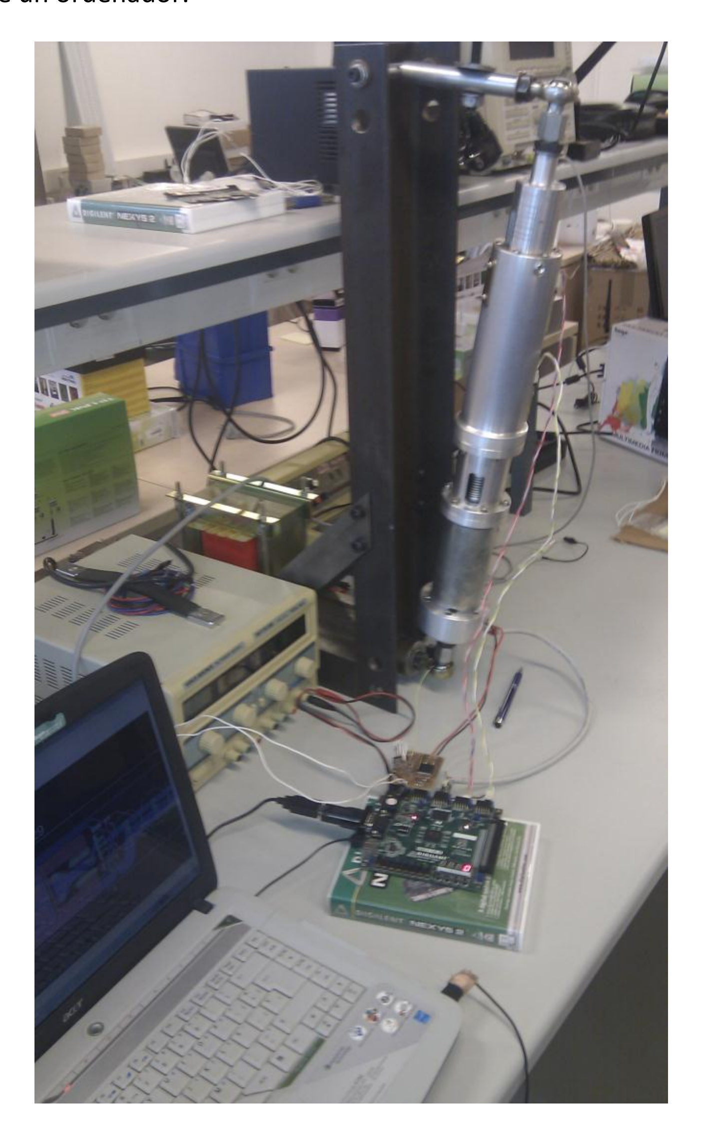
Para llevar a cabo el proyecto, se han completado los siguientes objetivos:

- Realización de un estudio previo del proyecto de partida, para su posterior modificación.
- Realización de un estudio previo del sistema físico a controlar, que engloba al encoder, motor y el resto del actuador lineal.
- Simulación del sistema físico y su estructura de control mediante simulink y matlab.
- Reestructuración de la placa de potencia, sustituyendo algunos de los componentes, limitando su tamaño y corrigiendo algunos errores.
- Montaje y verificación de la placa.
- Diseño digital en VHDL de la etapa de control en bucle cerrado y del envío y recepción de datos sobre una FPGA.
- Simulaciones del diseño para comprobar su correcto funcionamiento.
- Pruebas del motor con el encoder, primero fuera del actuador lineal, y luego con el sistema completo.
- Programa en matlab para el envío de una trayectoria sinusoidal y otra escalón, para permitir la caracterización posterior del actuador.

1.2 Motivación del proyecto.

Se busca la mejora de un proyecto ya hecho, cuya finalidad es la equinoterapia como tratamiento médico para pacientes con enfermedades neurodegenerativas y traumatológicas. Para más información véase beneficios de la equinoterapia en la memoria del proyecto *CONTROL CON FPGA DE ACTUADOR LINEAL PARA EMULADOR DE EQUINOTERAPIA de JAVIER MARCO ESTRUC.*

La finalidad del proyecto es que el actuador describa trayectorias prefijadas transmitidas desde un ordenador.



Motor en la estructura final.

En la imagen se observa la FPGA conectada al ordenador mediante su cable de alimentación y un cable adaptador USB-puerto serie (USB al ordenador y puerto serie a la FPGA). La placa marrón es la de potencia del proyecto anterior, que se conecta a la FPGA mediante un header, y del motor salen cuatro grupos de cables correspondientes a los dos finales de carrera y al encoder y los cables negro y rojo de la placa de potencia.

1.3 Diagrama de bloques del proyecto.

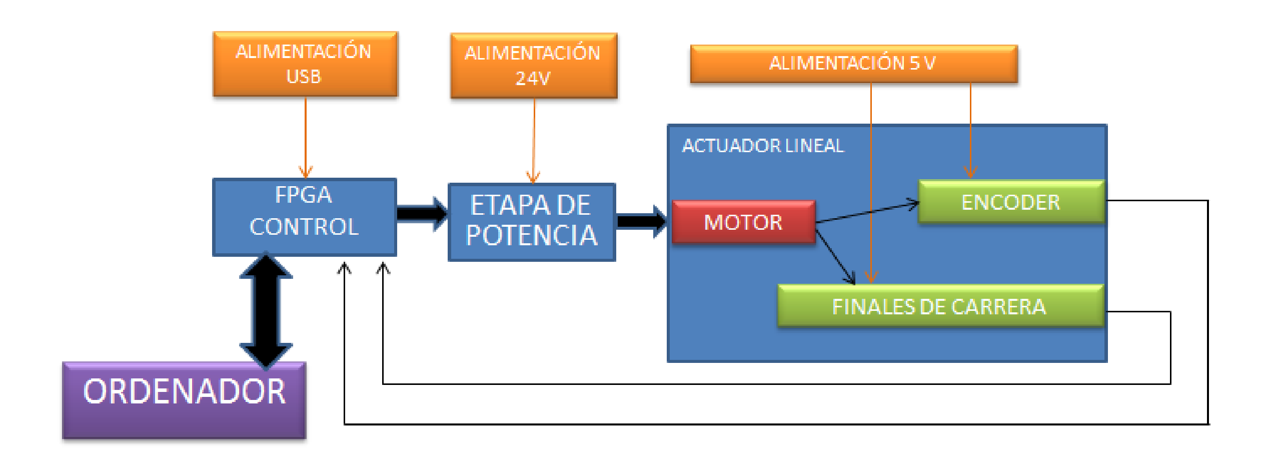


Diagrama de bloques general

Con el ordenador se envía a través de un programa realizado en MATLAB la trayectoria que el motor debe seguir. Para este proyecto se ha implementado una trayectoria senoidal y otra en escalón (Véase el capítulo de programación en MATLAB).

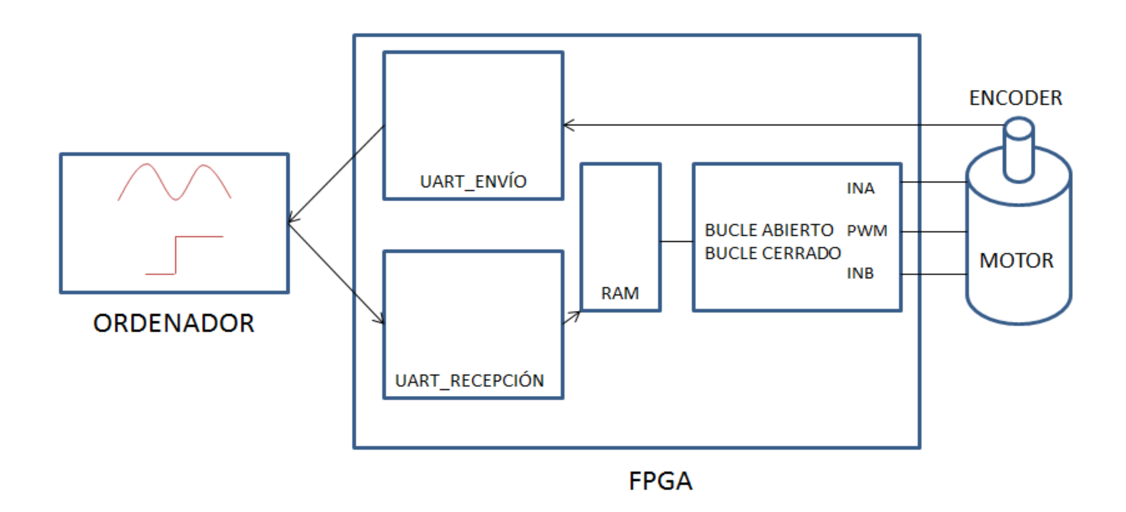
En la FPGA se ha diseño una UART de recepción para captar esta trayectoria. Los datos de la trayectoria son almacenados en una memoria RAM donde se almacenan hasta que recibe el último.

El actuador lineal tiene una carrera de 10 cm. y por criterio propio se ha elegido que la resolución entre posición y posición del mismo sea de una décima de milímetro. Por lo que los 10 cm. del vástago equivalen a 1000 posiciones del actuador.

El motor tiene solidario al eje un encoder que permite conocer la posición de éste.

Antes de realizar el control en bucle cerrado, hay que adaptar los datos de la trayectoria almacenada en la RAM, dados en décimas de milímetro, para poder comparar con la posición del encoder que viene dada en pulsos del encoder.

Finalmente para comprobar la respuesta del motor a la trayectoria enviada, también hay implementada una UART para enviar los datos del encoder desde la FPGA al ordenador, que también son procesados mediante MATLAB.



Esquema del proyecto.

1.4 Estructura del presente documento.

En este apartado se explica la estructura de la memoria, dando una pequeña introducción de cada uno de los apartados de los que consta.

La memoria está formada por 8 capítulos:

- Capítulo 1, introducción. En este apartado se da una visión general de los pasos dados en el desarrollo de este proyecto y de las tres grandes funciones que lo componen.
- Capítulo 2, control en bucle cerrado. Caracterización del sistema físico compuesto por el motor y su estructura y por el encoder, y de las inercias y rozamientos propios del sistema, así como de su control en bucle cerrado y de la estrategia seguida.
- Capítulo 3, diseño digital en VHDL.
- Capítulo 4, programación en MATLAB. Programación de dos trayectorias para el motor, una senoide y un escalón.
- Capítulo 5, conexión del actuador. Se detalla tanto el interfaz de usuario, como los pines de los finales de carrera y del encoder.
- Capítulo 6, simulaciones. Simulaciones del diseño en VHDL.
- Capítulo 7, PCB. Mejoras y reducción de la placa de potencia.
- Capítulo 8, anexos.

CAPÍTULO 2

2 Control en bucle cerrado

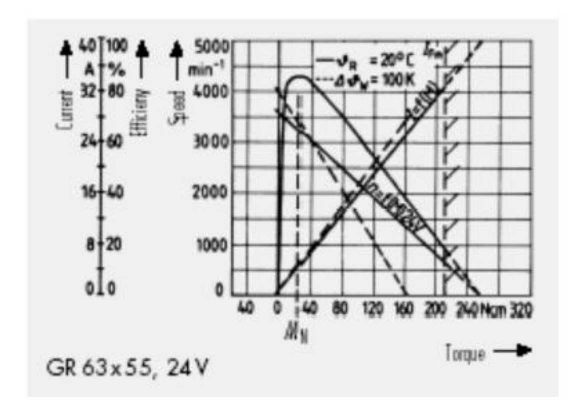
2.1 Motor de corriente continua.

El motor que se usa el modelo GR63x55 de ELMEQ que se caracteriza por ser de excitación independiente y tener imanes permanentes en vez de devanados.

Los principales motivos por los que se eligió fueron:

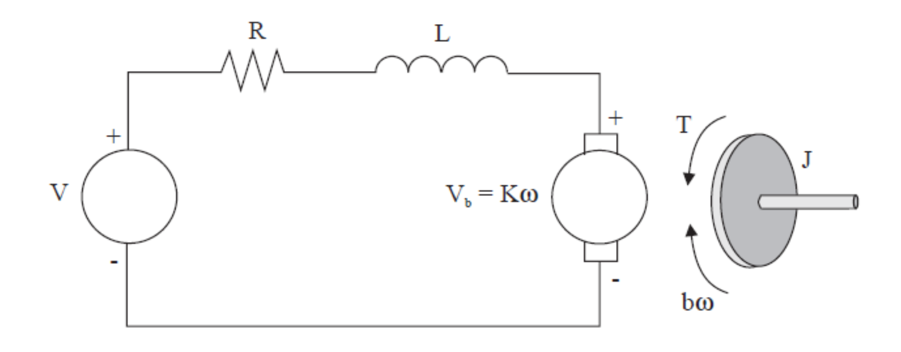
- Control muy sencillo.
- Requieren poca electrónica para su manejo.
- Elevada eficiencia.





Motor GR63x55 y curva característica.

2.2 Sistema físico.



Representación esquemática del motor DC.

Al aplicar una tensión en los bornes del motor, se genera una corriente en el devanado del inducido que al estar en un campo magnético generado por los imanes, producen una

fuerza proporcional a la corriente en los conductores que hace que el motor gire. Esta fuerza es el par motor. Representada en la figura como T (motor torque).

$$T = Ki\,.$$

donde K es la constante del Par Motor e i es la corriente del inducido.

Al mismo tiempo se genera una fuerza contraelectromotriz que se opone al giro del motor Vb en la figura.

$$V_b = K\omega = K\frac{d\theta}{dt}\,.$$

donde K es igual a la constante del par motor al ser un motor de imanes permanentes y ω es la velocidad angular.

Además el motor presenta otra serie de datos que han sido estimados por un alumno de ingeniería industrial y otros se han extraído de la hoja de características del fabricante:

```
k = 0.064;
L = 0.0015;
R = 0.6;
Jm = 0.000075;// Inercia del rotor del motor (Datasheet GR63X55)
Ja = 0.0002709;// Inercia de partes móviles del actuador respecto del motor (JMoranchel)
J = Jm+Ja;// Momento de inercia estimado de todo el actuador
tau_em=11e-2;% Constante electromecánica estimada de todo el actuador
b = J/tau_em;// Rozamiento
```

Datos utilizados en la simulación con scilab del motor y las partes móviles.

Combinando las leyes de Newton con las leyes de Kirchhoff:

$$\begin{aligned} J\frac{d^2\theta}{dt^2} + b\frac{d\theta}{dt} &= Ki, \\ L\frac{di}{dt} + Ri &= V - K\frac{d\theta}{dt}\,. \end{aligned}$$

Usando la transformada de Laplace:

$$Js^2\theta(s) + bs\theta(s) = KI(s), \quad (1)$$
$$LsI(s) + RI(s) = V(s) - Ks\theta(s) \quad (2)$$

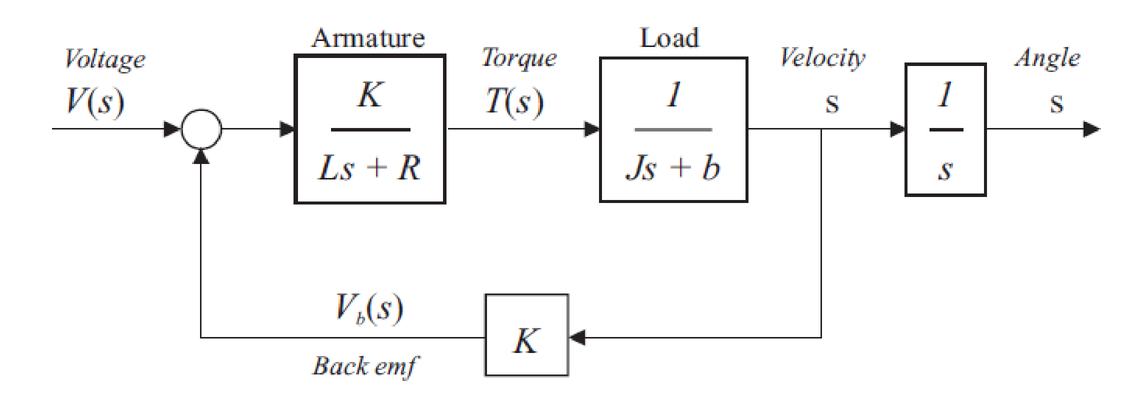
Despejando de (2):

$$I(s) = \frac{V(s) - Ks\theta(s)}{R + Ls}$$

Sustituyendo en (1):

$$Js^2\theta(s) + bs\theta(s) = K\frac{V(s) - Ks\theta(s)}{R + Ls}$$

El diagrama de bloques queda:



En función de la posición angular:

$$G_a(s) = \frac{\theta(s)}{V(s)} = \frac{K}{s[(R + Ls)(Js + b) + K^2]}$$

o de la velocidad angular:

$$G_v(s) = \frac{\omega(s)}{V(s)} = \frac{K}{(R + Ls)(Js + b) + K^2}$$

2.3 Bucle cerrado

Llegados a este punto, debido a que el encoder envía la señal de la posición en la que se encuentra el motor, tendremos que hacer un control de posición y no de velocidad.

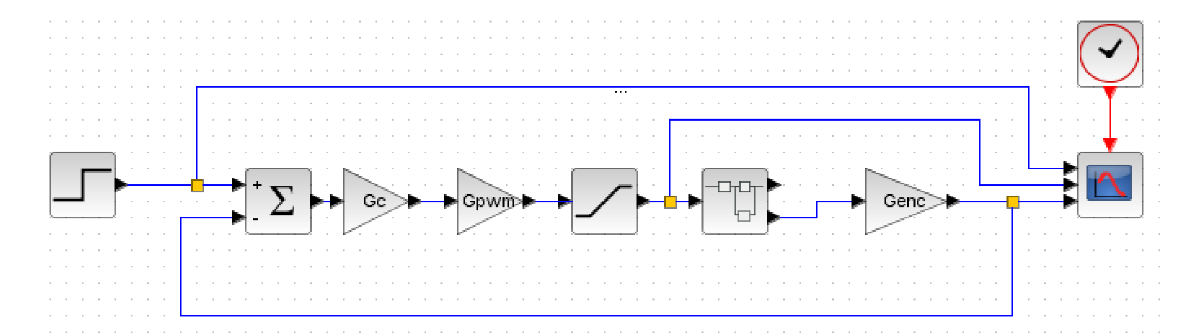
$$G_a(s) = \frac{\theta(s)}{V(s)} = \frac{K}{s[(R+Ls)(Js+b)+K^2]}$$

$$F(s) = \frac{0{,}064Kr}{0{,}000000518s^3 + 0{,}000212256s^2 + 0{,}0059827s + 0{,}064Kr}$$

Además debemos considerar otros elementos adicionales del sistema, como son:

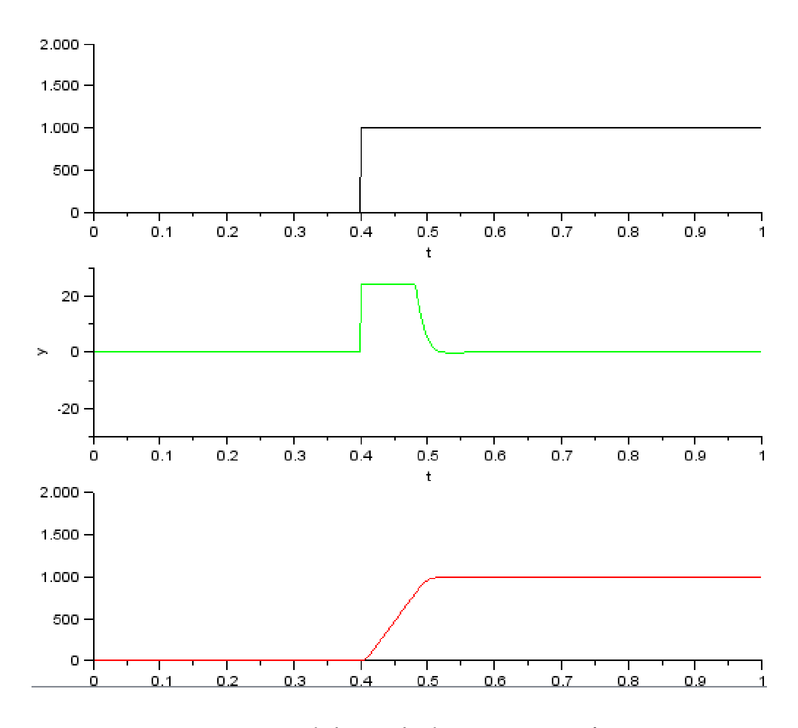
- Ganancia del encoder 360/(2*pi) ya que el encoder genera 360 pulsos para cada vuelta del motor.
- Saturación de +24 -24 voltios que es el máximo de la fuente de alimentación.
- Ganancia de la etapa PWM que cuenta de 0 a 100 para la alimentación de 0 a 24, por tanto, Gpwm=100/24.

Se ha simulado el sistema en bucle cerrado con un regulador proporcional, usando Scilab, para varias ganancias del regulador.



Esquema de simulación con Scilab.

Los resultados se muestran en las figuras siguientes, en las que se observa, en la primera gráfica, la entrada en escalón, en la segunda gráfica la tensión, y en la tercera la respuesta.



Ganancia del regulador 0,5 en vacío.

A la hora de hacer pruebas, se ha usado el motor en vacío por lo que los valores son los de la siguiente tabla.

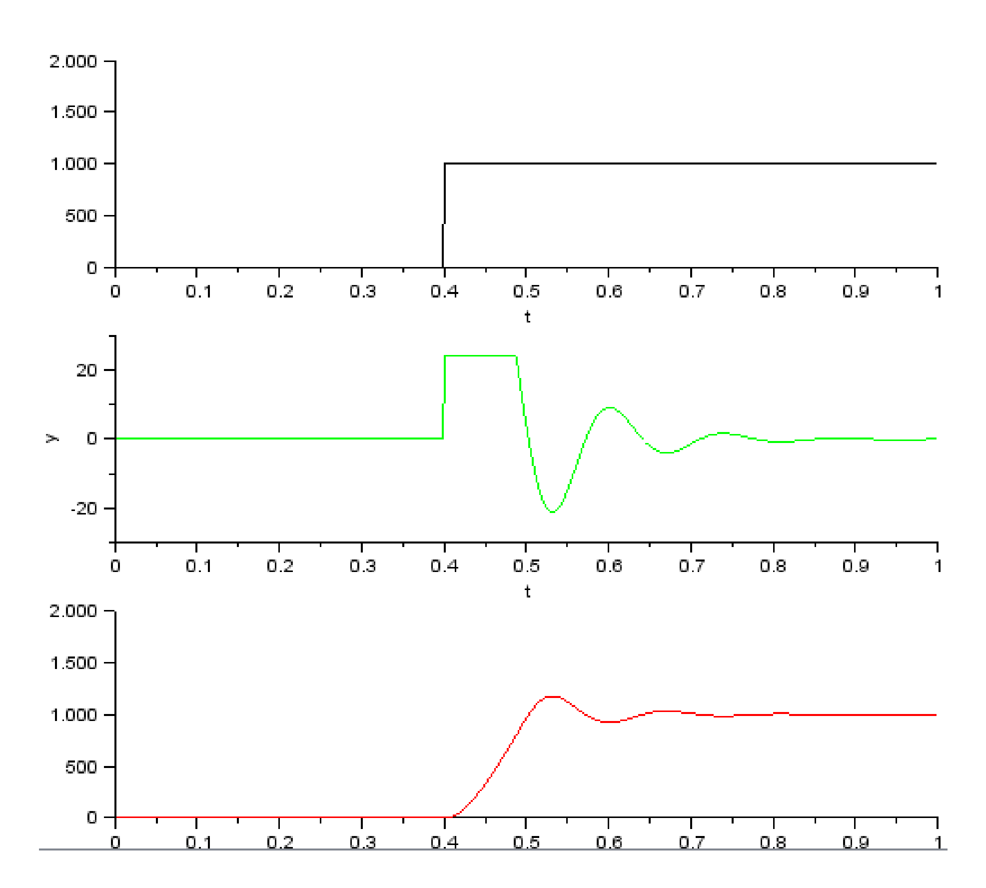
k = 0.064;
L = 0.0015;
R = 0.6;
Jm = 0.000075;*// Inercia del rotor del motor (Datasheet GR63X55)*
Ja = 0;*// Inercia de partes móviles del actuador respecto del motor (JMoranchel)*
tau_em = 0.011;// Constante de tiempo electromecánica del motor suelto, para estimar el rozamiento
J = Jm+Ja;*// Momento de inercia estimado de todo el actuador*
b = J/tau_em;*// Rozamiento*
Gpwm = 24/100;
Genc = 360/(2*%pi);*// Ganancia del encoder (pulsos/rad)*
Gc = 0.5;

Datos utilizados en la simulación con scilab en vacío.

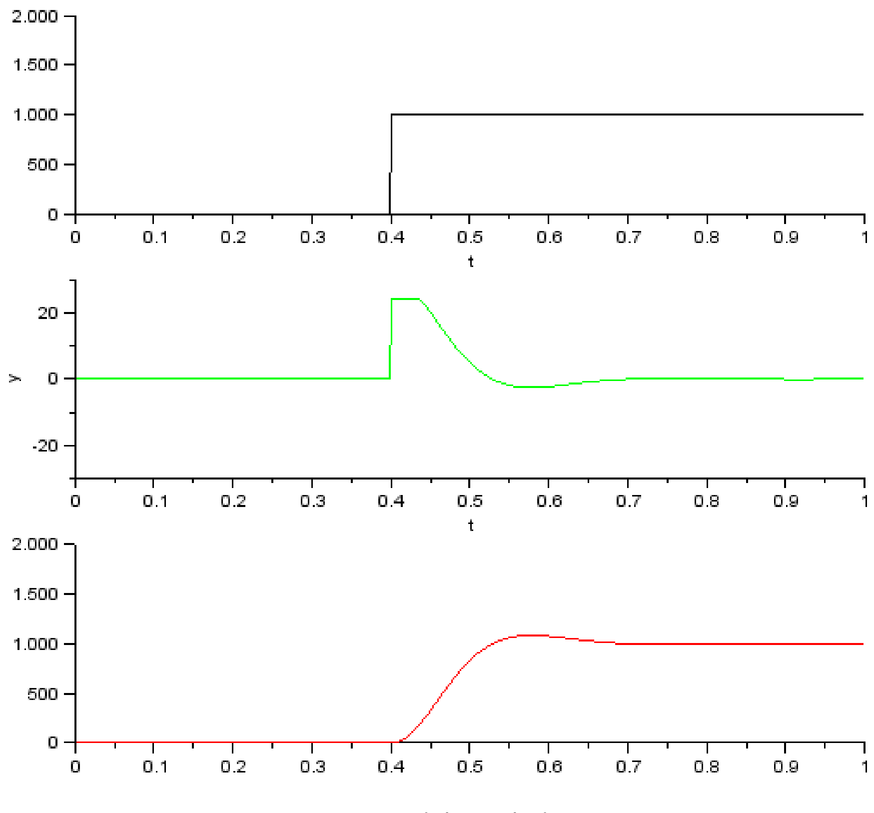
Para el actuador los valores son los siguientes.

k = 0.064;
L = 0.0015;
R = 0.6;
Jm = 0.000075;*// Inercia del rotor del motor (Datasheet GR63X55)*
Ja = 0.0002709;// Inercia de partes móviles del actuador respecto del motor (JMoranchel)
J = Jm+Ja;*// Momento de inercia estimado de todo el actuador*
tau_em=11e-2;% Constante electromecánica estimada de todo el actuador
b = J/tau_em;*// Rozamiento*
Gpwm = 24/100;
Genc = 360/(2*%pi);*// Ganancia del encoder (pulsos/rad)*
Gc = 0.5;

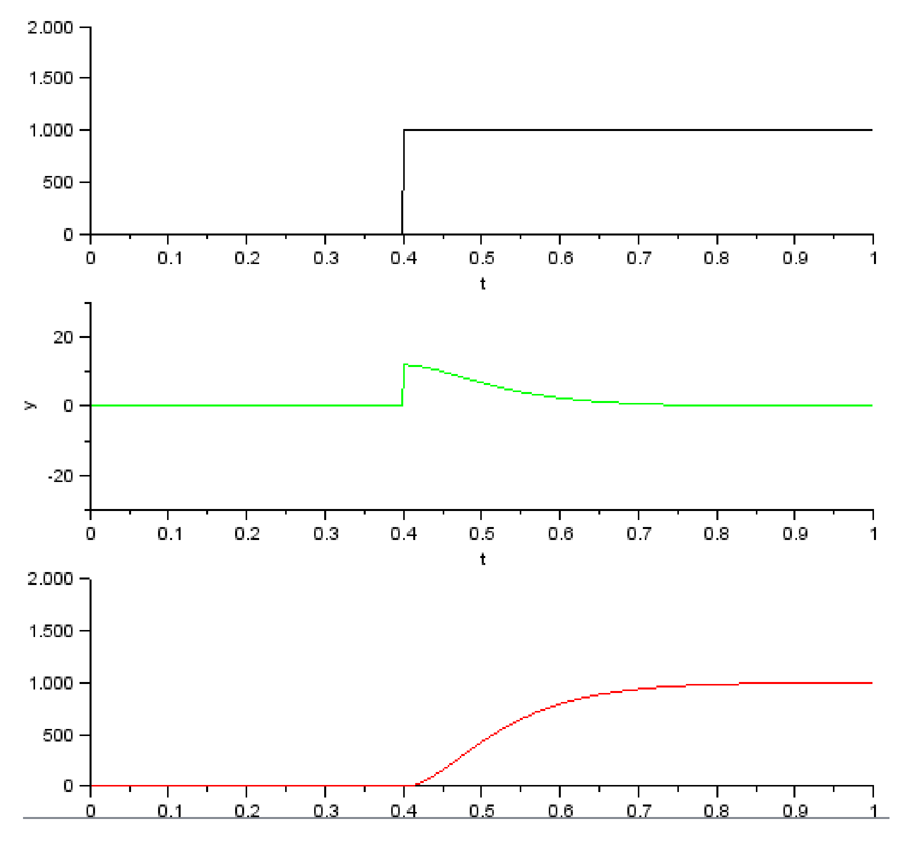
Datos utilizados en la simulación con scilab con el actuador.



Ganancia del regulador 0,5 con sobreoscilación en la respuesta.




Ganancia del regulador 0,125.



Ganancia del regulador 0,05.

CAPÍTULO 3

3 Diseño digital en VHDL.

3.1 Introducción.

A la hora de abordar el diseño en VHDL se ha dividido la estructura en 5 grandes bloques:

- UART emisor, envía la posición del encoder al ordenador cada 10 milisegundos.
- UART receptor, recibe la trayectoria (senoidal o escalón) del ordenador.
- Memoria RAM, almacena la trayectoria recibida.
- Adaptación de los datos almacenados en la RAM al bucle cerrado.
- Control bucle cerrado, permite que el actuador siga la trayectoria almacenada en la RAM mediante las tres salidas INA, INB y PWM.

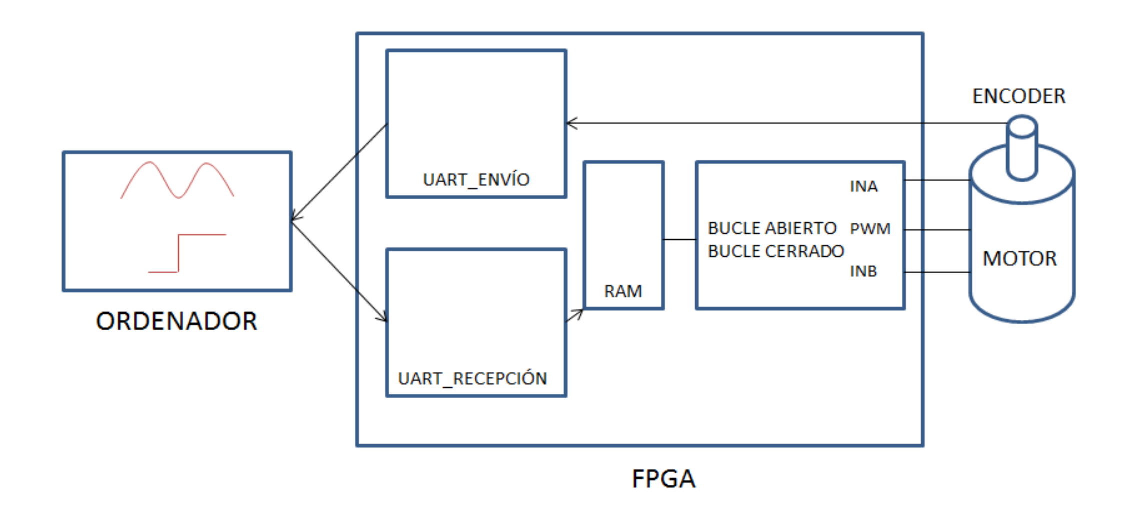
A parte de sus respectivas máquinas de estado y de la programación en VHDL ya implementada.

La trayectoria a seguir por el actuador se envía discretizada con una frecuencia de muestreo de 100 Hz y cuantificada en décimas de milímetro, es decir, si el recorrido máximo del actuador lineal es de 10 cm. significa que son 1000 posiciones las que se puede abarcar. Estas 1000 posiciones se codifican en 10 bits (2^{10}=1024) y el envío de datos se realiza de 8 en 8 bits. Como los 10 bits no pueden ser enviados juntos, se separan en 2 vectores de 5 bits cada uno y se envían como los bits menos significativos. Los otros 3 que sobran se utilizan para mandar instrucciones al motor, desde realizar una parada, hasta volver a enviar los datos. Esto permiterealizar el interfaz de usuario desde el ordenador, sin tener que interactuar con las teclas de la placa de la FPGA.

Los datos de la trayectoria se almacenan en memoria RAM dentro de la FPGA, cada dato en dos vectores de 5 bits y cuando se inicia el movimiento del actuador van introduciéndose como consigna del bucle cerrado cada 10 ms.

Para el control en bucle cerrado se compara la posición del motor que se recibe del encoder, con la posición a la que tiene que estar el motor, que ha sido almacenada en la RAM. Como la posición viene en décimas de milímetro y la información del encoder en pulsos, hace falta su adaptación, sabiendo que una décima de milímetro equivale a 12 pulsos.

Del control en bucle cerrado se obtienen las salidas INA, INB y PWM. INA para el sentido de subida, INB para el sentido de bajada y PWM para la velocidad.



3.2 UART emisor.

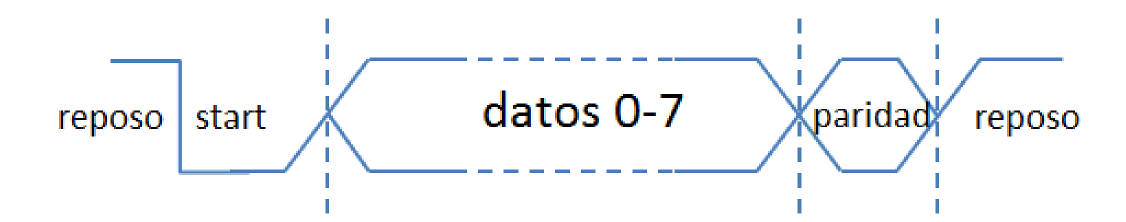
Este diseño ha sido extraído del libro ELECTRÓNICA DIGITAL APLICACIONES Y PROBLEMAS de José Ignacio Artigas, Luis Ángel Barragán, Carlos Orrite e Isidro Urriza referenciado en la bibliografía del proyecto.

El bloque emisor es el encargado de recoger la palabra de 8 bits y transmitirla por la línea serie *Tx* siguiendo el protocolo serie asíncrono. El conjunto de señales que llegan y salen de este bloque aparece descrito en la tabla siguiente:

Señal	Dirección	Descripción
CLK	Entrada	Reloj
EN	Entrada	Habilitación del reloj de entrada
RST	Entrada	*Reset*
WR	Entrada	Habilitación para escribir un nuevo dato en el emisor
DIN	Entrada	Dato en paralelo para ser transmitido
Tx	Salida	Transmisión serie
TxRDY	Salida	Listo para enviar un nuevo dato

Señales de entrada/salida del emisor.

Tanto emisor como receptor van a trabajar con este formato de envío/recepción de datos:



El emisor se compone de varios bloques: un controlador basado en una máquina de estados, un contador *CNTBIT* que lleva la cuenta del bit de datos transmitido, un registro de desplazamiento paralelo/serie y un generador de paridad impar.

En lo que respecta al controlador, podemos distinguir cuatro estados en la transmisión. En un primer estado, que denominaremos *STOP*, el emisor permanece en reposo a la espera de un nuevo dato a transmitir. La llegada de dicho dato se informa al emisor mediante la activación de la señal *WR*, momento en el que pasamos al estado *INICIO*, donde el *byte* presente en *DIN* pasa al registro de desplazamiento. Tras este estado pasamos a *DATOS*, donde se enviarán en serie los 8 bits. Un contador *CNTBIT* determina qué bit se transmite por *Tx*. Una vez se ha enviado el bit más significativo, pasamos al estado *PARIDAD* donde se transmite el bit de paridad del conjunto, para terminar con el envío del bit de stop, en el estado *STOP*. Obsérvese que el estado *STOP* corresponde tanto al envío del bit de stop como a la situación de reposo a la espera de enviar un nuevo dato.

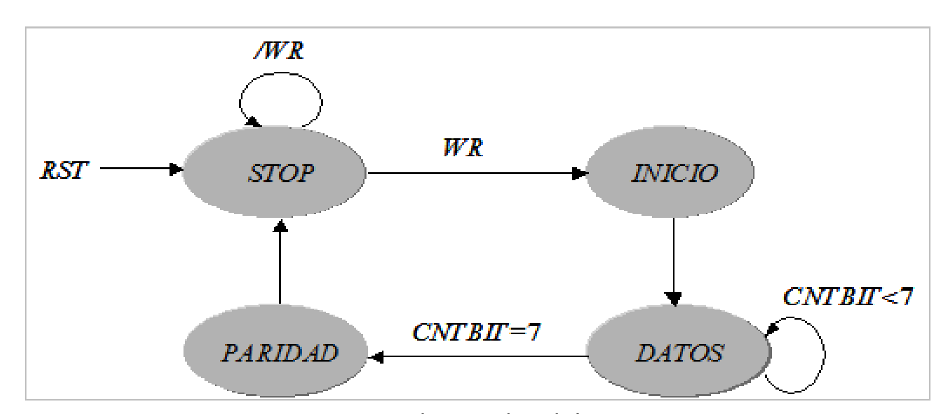


Diagrama de estados del emisor.

Para controlar el envío de datos desde el programa principal, se ha implementado otra máquina de estados con cuatro estados:

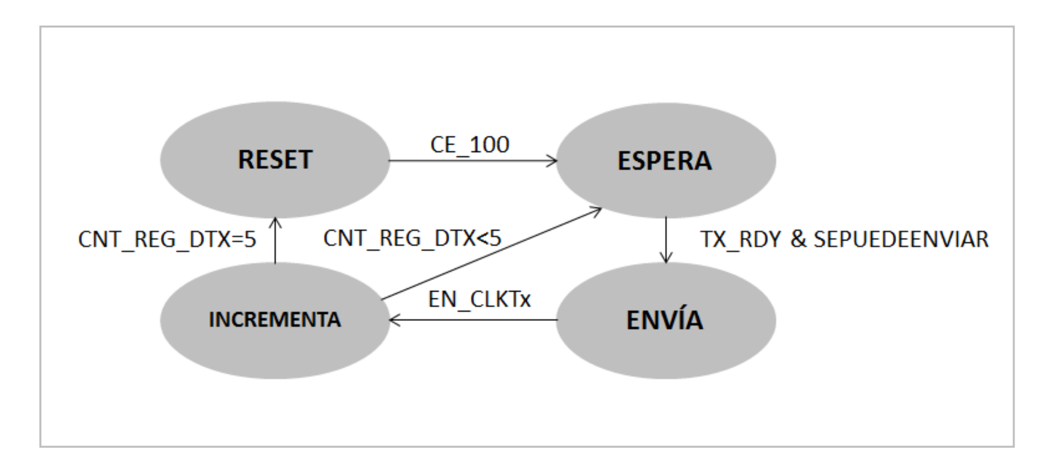


Diagrama de estados del control del emisor.

Se dispone de 5 registros que abarcan desde REG_DTX (0) hasta REG_DTX (5) que se han utilizado para diferentes pruebas en el envío de datos. Cuando la UART está disponible (se indica con la activación de TX-RDY) se manda el primer registro, se incrementa el contador en el estado INCREMENTA, que hace referencia al número del registro, y se envía el siguiente registro, hasta completar el envío de los 5. Momento en el cual se repite el ciclo.

CE_100 se activa cada 100 Hz, lo que implica que este bucle se ejecuta cada 10 ms todo el tiempo, por lo que se ordena con SEPUEDEENVIAR si se quiere enviar o no.

3.3 UART receptor.

El bloque receptor recibirá el dato por *Rx* y, una vez convertido en paralelo, pasará a la salida *DOUT*, indicándose mediante la activación de la señal *RxRDY*.

Se compone de varios bloques: sincronizador de la línea de entrada; máquina de estados; contador de reloj que permita situarnos en el punto medio del nivel de la señal de recepción; contador del bit de dato; generador de paridad; registro de desplazamiento serie/paralelo; generación de salida de datos y señales referentes al estado de la comunicación.

En la siguiente tabla se muestra el conjunto de señales de entrada/salida de este bloque.

Señal	**Dirección**	**Descripción**
CLK	Entrada	Reloj
EN	Entrada	Habilitación del reloj de recepción
RST	Entrada	*Reset*
Rx	Entrada	Recepción serie
RD	Entrada	Lectura del dato recibido
DOUT	Salida	Dato en paralelo recibido
RxRDY	Salida	Indica que un nuevo dato ha sido recibido y está listo para ser leído
ERROR_PARIDAD	Salida	Error en paridad
ERROR_FORMATO	Salida	El formato no se corresponde con el RS232
SOBRESCRITURA	Salida	Ha llegado un dato sin haber leído el anterior.

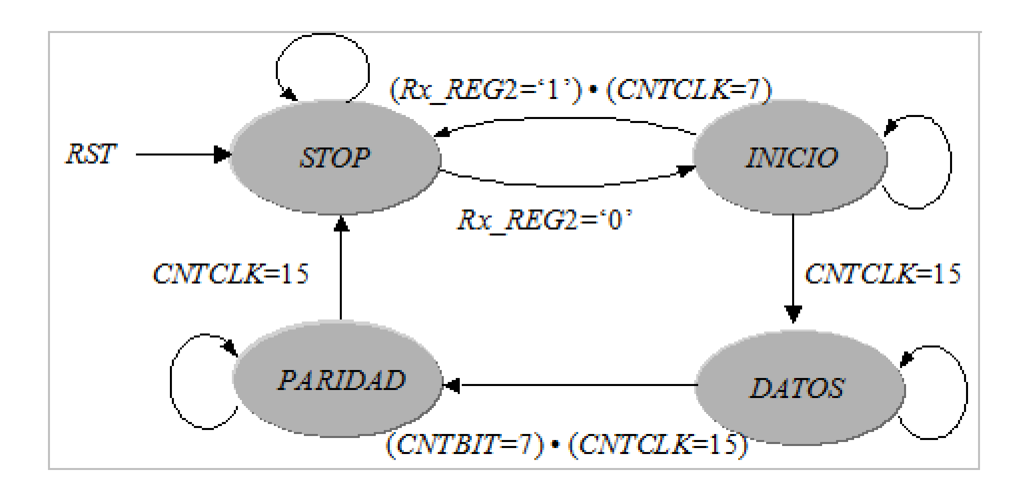
Señales de entrada/salida del bloque receptor.

Al ser una señal asíncrona, el primer bloque será un sincronizador compuesto por dos biestables: *Rx_REG* y *Rx_REG2*, en cascada.

Al igual que hemos visto en la transmisión, en recepción podemos distinguir también cuatro estados. Un primer estado de reposo que denominaremos *STOP*, en el que el receptor espera la llegada de un nuevo dato por la línea serie. Al detectar el flanco de bajada, pasamos al estado *INICIO*, donde se procede a la sincronización. Para ello, utilizamos el contador *CNTCLK* que nos permita leer en el punto medio del bit recibido. Cada bit recibido equivale a 16 pulsos de la señal de habilitación del reloj de recepción *EN_CLKRx*; por tanto, debemos situarnos en el pulso octavo del reloj y, a partir de aquí, cada dieciséis pulsos de reloj leer la línea serie. Al situarnos en el octavo pulso en el estado *INICIO* comprobaremos de nuevo si la señal *Rx_REG*2 permanece en nivel bajo. Si no es así, supondremos que la comunicación ha sido generada por ruido en la línea y volveremos al estado *STOP*.

Tras el estado *INICIO*, pasamos al estado *DATOS* donde leeremos el bit presente en *Rx* y lo iremos colocando en el registro de desplazamiento interno. Para llevar la cuenta de los bits de datos recibidos, utilizaremos el contador *CNTBIT*, únicamente habilitado en este estado. Cuando dicho contador alcanza el valor 7, el siguiente bit corresponderá al bit de paridad transmitido. En este momento ya tendremos calculada la paridad del *byte* recibido y podremos comparar ambos valores, cargando en el registro de salida *DOUT* el dato desplazado si éstas son iguales, al tiempo que activaremos la señal *RxRDY*. El último bit del protocolo RS232, debe

ser un nivel alto, equivalente al valor que toma la línea serie cuando el sistema permanece en reposo sin envío de datos. Por ello, se ha elegido un único estado, denominado *STOP*, que represente ambas situaciones: correspondiente al bit de stop y al estado de reposo.



El bloque receptor genera tres señales de salida que permiten comprobar el estado de la comunicación. La señal *ERROR_PARIDAD* indica al receptor de la recepción errónea de un dato al comprobar la paridad:

```
ERROR_PARIDAD <= '1' when (ESTADO=PARIDAD and CNTCLK=7 and BIT_PARIDAD/=Rx_REG2) else
'0';
```

La señal *ERROR_FORMATO* indica que el formato de recepción no se corresponde con el RS232. Una forma fácil de comprobarlo consiste en verificar que el bit de *STOP* sea '1':

```
ERROR_FORMATO <= '1' when (ESTADO=STOP and CNTCLK=7 and Rx_REG2='0') else '0';
```

Por último, la señal *SOBRESCRITURA* indica si se ha recibido un nuevo dato sin haber llegado a leer el dato previamente recibido:

```
SOBRESCRITURA <= '1' when (ESTADO=PARIDAD and CNTCLK=7 and IGUAL_PARIDAD='1' and
RxRDY_REG='1') else '0';
```

Aunque estas señales no se usan.

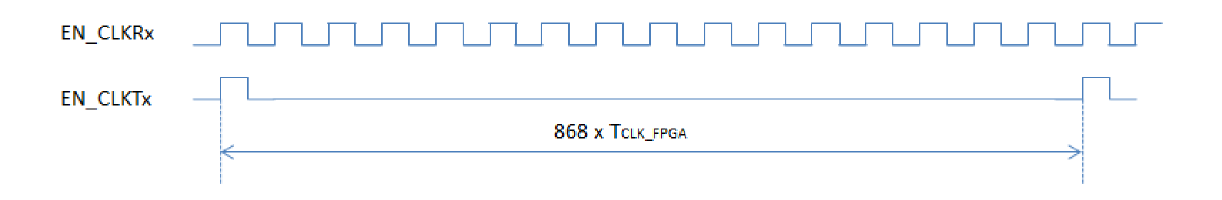
El registro que almacena el dato recibido en el programa principal es DRX. No hace falta una máquina de estados adicional en el programa principal.

3.4 Habilitación del reloj del emisor y receptor.

La velocidad de recepción y envío de datos se ha elegido de 57600 baudios, por lo que si la FPGA funciona a 50 MHz:

$$\frac{50.000.000}{57.600} = 868$$

868 para el emisor, ya que el receptor tiene que producir 16 pulsos por cada bit lo que implica que tiene que ser 16 veces más rápido que el emisor.



Por lo tanto, partiendo de la habilitación del reloj de recepción que es la más rápida, se genera la del emisor.

$$\frac{868}{16} = 54$$

Con un contador de que cuenta de 0 a 53 se crea la habilitación del receptor y con otro de 16 la del emisor.

Las operaciones anteriores no son exactas, por lo que se produce un error debido al redondeo. Esta es la velocidad a la que se programa:

$$\frac{50.000.000}{16*54} = 57.870 \text{ b/s}$$

Cuya diferencia con la anterior es de 0,081µs por bit.

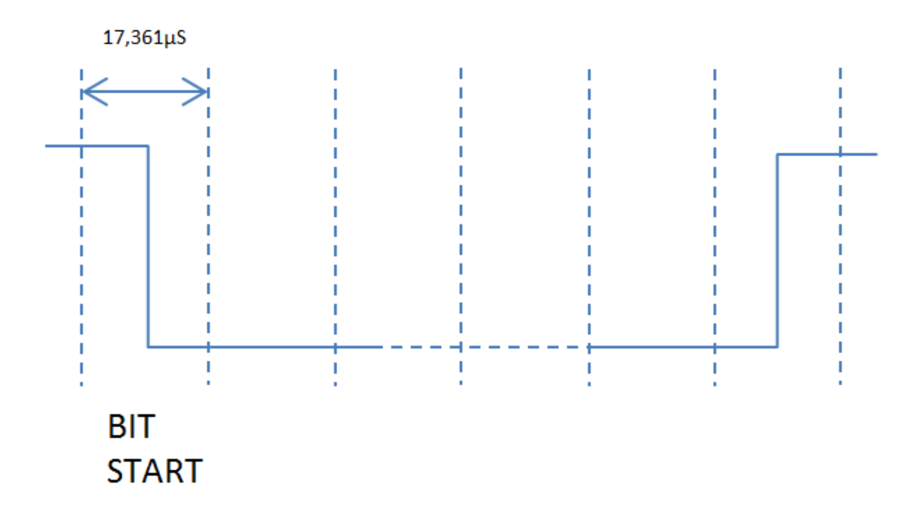
$$\frac{1}{57.870 b/s} = 17{,}280\mu\text{s} \qquad \frac{1}{57.600\ b/s} = 17{,}361\mu\text{s}$$

$$17{,}361\mu\text{s} - 17{,}280\mu\text{s} = 0{,}081\mu\text{s}$$

Cada serie de datos está compuesta por 11 bits. 8 bits del dato, start, paridad y reposo. Por lo que se puede producir un desfase de:

$$0{,}081\mu s * 11 = 0{,}9\mu s$$

Lo cual comparado con el tiempo de un bit (17,361 µs) no supone ningún problema a la hora de hacer coincidir los bits con el tiempo de bit.



3.5 Memoria RAM.

En la memoria RAM se guardan todos los valores recibidos a través de la UART.

La resolución con la que se trabaja es de 0.1 mm. y la carrera del actuador lineal es de 10 cm. lo que significa que hay 1000 posiciones intermedias a lo largo del vástago con esa resolución. Es decir, para codificar estas 1000 posibles posiciones se necesita un vector de ese tamaño o mayor. Con $2^{10} = 1024$ se cumple esta condición.

El dato recibido a través de la UART es de 8 bits, los tres más significativos son utilizados para enviar las instrucciones al actuador. Los 5 restantes son los que contienen el dato. Por lo que éste se recibe en dos partes, primero los 5 bits más significativos y después los menos significativos.

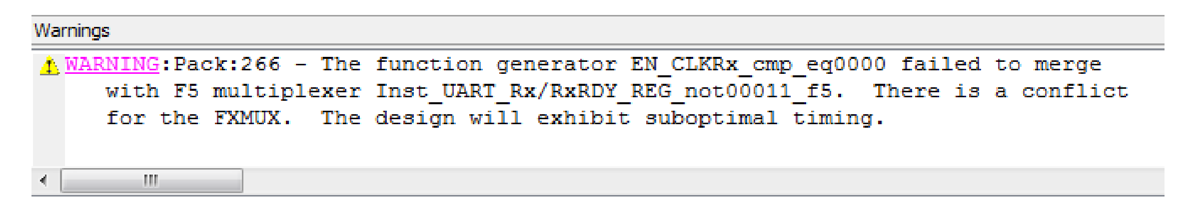
La memoria RAM se estructura en bloques de 5 bits, cada dos se tiene un dato completo.

Cada 10 milisegundos se lee un dato completo de la RAM (10 bits). Con un tiempo elegido de un minuto para realizar un movimiento completo del actuador, se tiene:

$$\frac{60\ segundos}{10\ milisegundos} = 6000\ posiciones$$

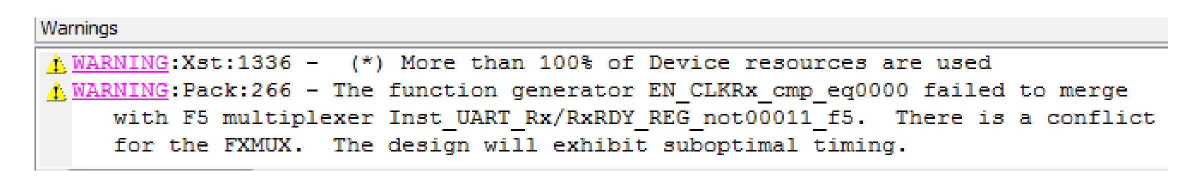
6000 posiciones y cada posición son 2 vectores de 5 bits, esto implica que son necesarios 12.000 vectores. La potencia de dos más cercana a este número y por encima es 14 (2^{14}=16.384).

Con la FPGA usada con 2^{10} da el siguiente error:


Warnings

WARNING:Pack:266 - The function generator EN_CLKRx_cmp_eq0000 failed to merge with F5 multiplexer Inst_UART_Rx/RxRDY_REG_not00011_f5. There is a conflict for the FXMUX. The design will exhibit suboptimal timing.

Pero como el motor trabaja a una frecuencia baja no hay problema. Sin embargo a partir de 2^{13} la FPGA se queda pequeña y no es capaz de almacenar los datos.


Warnings

WARNING:Xst:1336 - (*) More than 100% of Device resources are used

WARNING:Pack:266 - The function generator EN_CLKRx_cmp_eq0000 failed to merge with F5 multiplexer Inst_UART_Rx/RxRDY_REG_not00011_f5. There is a conflict for the FXMUX. The design will exhibit suboptimal timing.

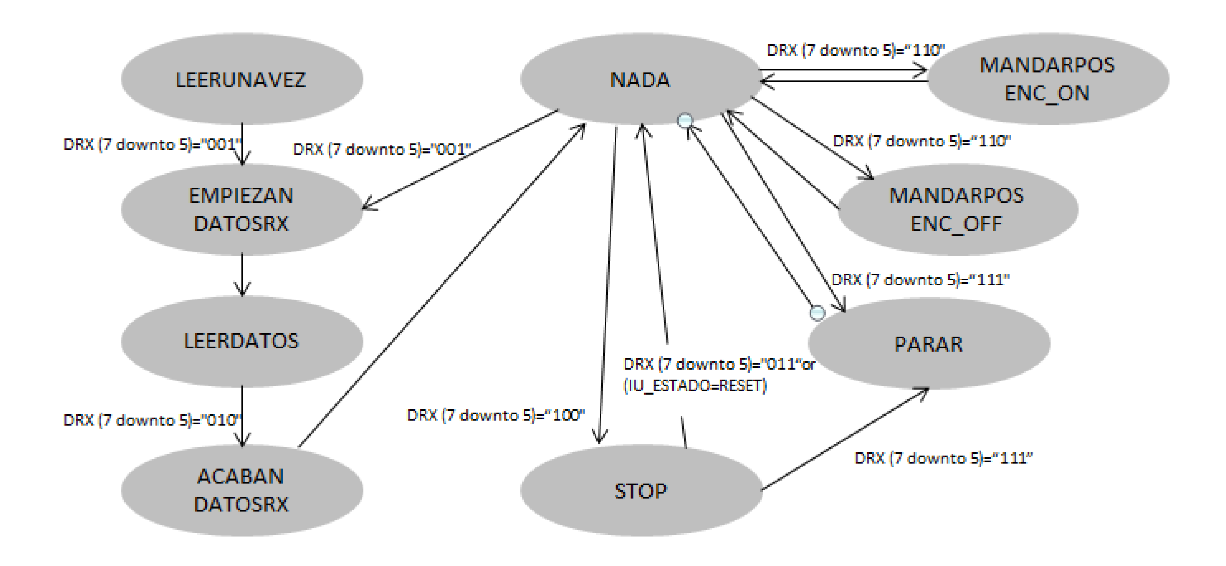
RAM con 2^{13}

Por eso en este proyecto se trabaja con 2^{12}, porque para minimizar el tamaño que ocupa hacen falta herramientas de las que no se dispone. 2^{12} equivale a 4096 vectores y se necesitan 2 vectores para formar un dato. El máximo de datos que se puede almacenar con este tamaño es de 2048.

Para acceder a la memoria se dispone de dos contadores, CNT_WRITE_RAM y CNT_READ_RAM, que como indica su nombre uno es para escribir en la Ram y otro para leer de la Ram.

3.6 Máquina de estados para el control del motor según las instrucciones recibidas.

Éste apartado junto con el siguiente son los más importantes y explican el funcionamiento del motor desde una perspectiva a nivel usuario y desde dentro.



Llegados a este punto, se plantea el control del motor a nivel de usuario (UART_ESTADO).

Empieza en el estado LEERUNAVEZ porque si no hay datos el motor no puede moverse, una vez introducidos los datos, tras haber pasado por los estados LEERUNAVEZ, EMPIEZANDATOSRX, LEERDATOS y ACABANDATOSRX, se habrá leído la trama de datos y no se volverá al estado LEERUNAVEZ. El siguiente estado es NADA, donde no se manda ninguna instrucción.

Existen otras condiciones que no aparecen en la imagen para no complicar el esquema que son:

Si FCI='0' o FCS2='0' es decir, el motor llega al final de carrera superior o inferior y el estado de control del motor es AJUSTAR o CONTROL (se explica en el siguiente apartado), es decir el motor está en movimiento el siguiente estado de control sea STOP, esto lo que permite es parar en motor en bucle cerrado, o sea que no avance más debido a las inercias propias del motor. En el estado BAJAR_TOPE también se ha implementado pero sólo para el final de carrera superior, por que el inferior es utilizado para la transición al siguiente estado.

En el apartado anterior a la hora de recibir datos, sólo 5 de los 8 bits recibidos eran utilizados. Por ello en este otro apartado se aprovecha los tres bits más significativos para recibir las instrucciones, que son las siguientes:

- “000” no pasa nada, sirve para borrar el registro DRX, debido a que este registro no cambia hasta no recibir un nuevo dato.

Se da el supuesto de que por ejemplo, al llegar al estado STOP a causa de un final de carrera (el caso comentado anteriormente), si ha habido un estado STOP-START antes, lo último que el registro DRX contiene es “011” es decir, se le manda que pare (estado STOP) pero al mismo tiempo, como no se ha borrado el registro DRX, se le manda que continúe, lo que debido a la frecuencia a la que trabaja la FPGA, al ojo humano es como si no parase. Por lo tanto en el programa de MATLAB en el momento que se envía “011” (START) inmediatamente después se envía “000” para limpiar el registro y así evitar problemas.

- “001” enviar, empieza la trama de datos a recibir. Los 5 bits menos significativos enviados junto a “001” también son válidos, se guardan en la memoria RAM.

- “010” terminar, la trama de datos finaliza. Los 5 bits menos significativos enviados junto a “010” no son válidos, no se guardan en la memoria RAM.

- “011” start, sólo puede darse tras el estado STOP, si se envía, pero no estaba previamente en ese estado, no pasa nada.

- “100” stop, para el motor pero el control en bucle cerrado sigue, esto se consigue manteniendo la misma consigna de posición todo el tiempo. El siguiente estado sólo puede ser el de continuar (con “011”) o el de PARAR.

- “111” parar, el estado PARAR es sólo una transición que permite cambiar al estado RESET del motor. Está pensado para cuando el estado STOP es debido a que se llega a un fin de carrera, si no existiera, después del STOP vendría el START y el motor seguiría realizando la trayectoria que tenía antes de parar, pudiendo sobrepasar los finales de carrera. Para ello es necesaria la alternativa de PARAR definitivamente, que permite empezar en el estado RESET sin continuar el movimiento.

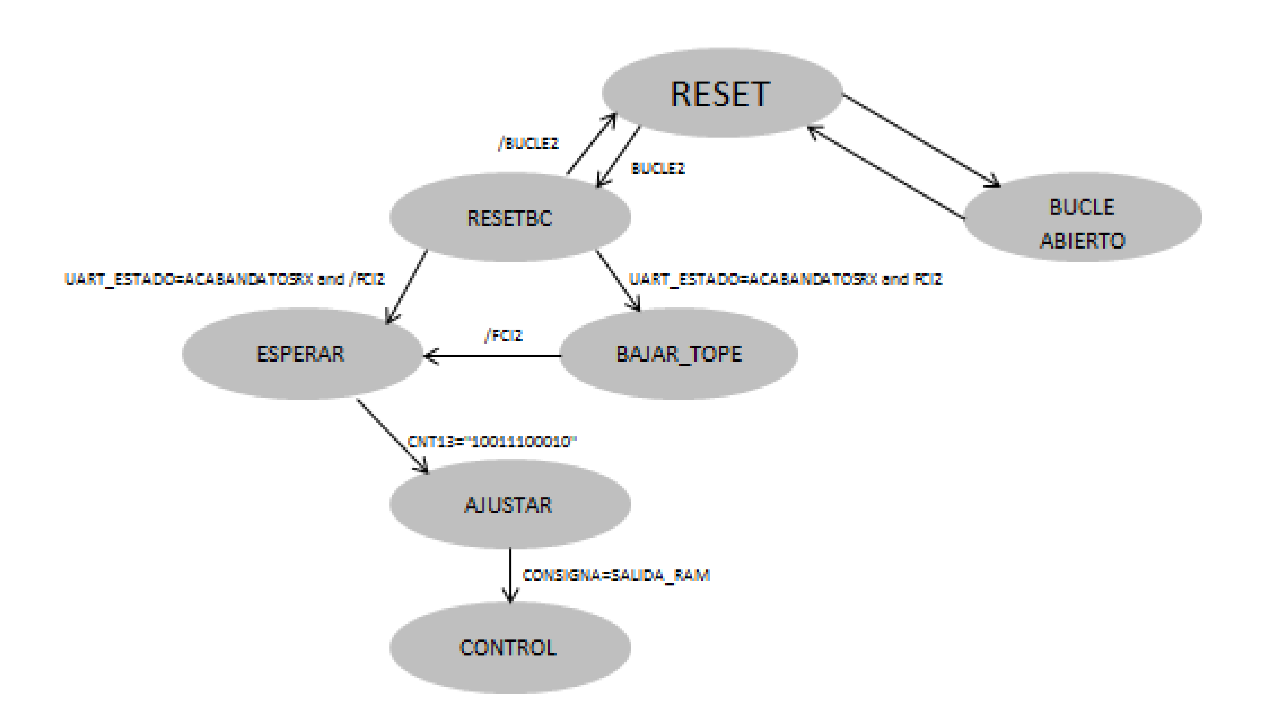
Los bits más significativos “111” se envían justo antes de mandar una nueva trama de datos en el programa de MATLAB. Desde que el motor alcanza el final de carrera hasta que el usuario introduce la nueva trama de datos, el motor ya ha parado.

- “101” mandar posición del encoder. Por defecto está activada.

- “110” no mandar posición del encoder. No se utiliza. Debido a que cada 10 milisegundos se recibe la posición del encoder al ordenador y a que se han utilizado programas que los recibían constantemente, se pensó en la posibilidad de anular el envío. Finalmente con la utilización de MATLAB de que sólo recibe los datos cuando hace la llamada de lectura del puerto, no se ha utilizado pero se ha dejado.

3.7 Máquina de estados del motor según si es bucle abierto o bucle cerrado.

Se ha realizado una máquina de estados (IU_ESTADO) para el control de la posición de referencia en bucle cerrado y para la salida PWM e INA e INB.



Donde el control el bucle abierto ya venía implementado del proyecto anterior.

Los estados BAJAR_TOPE, ESPERAR, AJUSTAR y CONTROL además tienen otras condiciones comunes que no se contemplan en la máquina de estados, para no complicar el dibujo, que son:

Si ILIM2 es '0' entonces nIU_ESTADO será RESET, es decir que si hay una sobrecorriente, no se siga con el programa.

Si UART_ESTADO=PARAR entonces nIU_ESTADO será RESET. Esta condición se cumple cuando FCI y FCS se ponen a '0'y permite que el motor se pare en bucle cerrado, o lo que es lo mismo, no permite que el motor se pase del final de carrera debido a la inercia propia de éste.

Si BUCLE2='0' esto significa que queremos acabar con el bucle cerrado, para un control por bucle abierto, y por tanto el siguiente estado es RESET.

Un switch permite elegir entre control en bucle cerrado y bucle abierto, partiendo siempre del mismo estado de inicio RESET.

En el estado RESETBC se está a la espera de recibir la trama de datos que contienen las posiciones que marcan la trayectoria que debe seguir el motor. Y hasta que no se recibe el último dato no se puede cambiar de estado.

Dependiendo de si el motor se encuentra abajo del todo FCI='0' o de si está en alguna posición intermedia o arriba, el siguiente estado será ESPERAR si ya está abajo, o BAJAR_TOPE si no lo está.

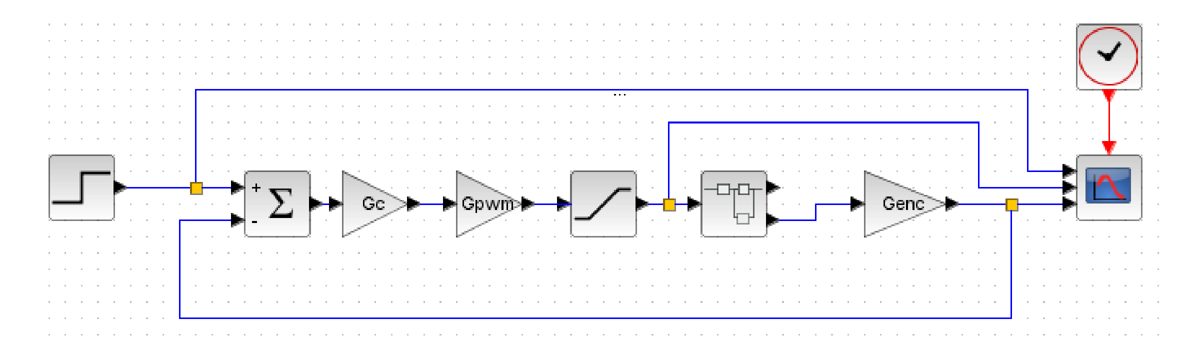
El estado ESPERAR se ha implementado con la idea de que el control sirva para más de un motor, es decir, dado el caso de tener dos motores o más, si uno está en una posición intermedia y otro está en una posición superior o inferior, el primero que llegue abajo tendrá que esperar al otro para empezar el siguiente estado AJUSTAR a la vez. Por eso se ha creado un contador, que tras 13 segundos (la explicación de porqué este tiempo y no otro se da más adelante), permita la transición de un estado a otro.

En el estado AJUSTAR sólo es necesaria la primera posición de la trayectoria que sigue el motor, que tras un tiempo es alcanzada por el motor a una velocidad constante.

En el estado CONTROL se envía toda la trama de datos almacenada en la memoria RAM para que el motor la siga.

3.8 Preparación del dato de referencia desde la memoria RAM hasta el bucle cerrado.

En el apartado 2 de esta memoria, el sistema físico se representaba en este diagrama de bloques:



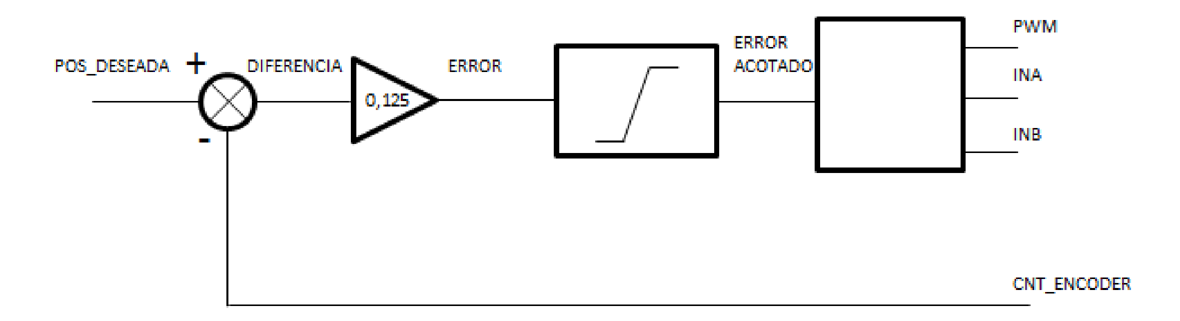
Donde el control se realiza mediante programación.

A la FPGA llega la información del encoder que se guarda en un contador llamado CNT_ENCODER. Este contador se incrementa o decrementa dependiendo del sentido de giro del motor.

```
elsif (PULSO_CHA='1') then
  if (CHB3='0') then
    nCNT_ENC <= CNT_ENC + 1; --sube
  else
    nCNT_ENC <= CNT_ENC - 1; --baja
  end if;
end if;
```

Y de la FPGA salen tres señales, que son INA que indica que el motor tiene que girar para subir, INB que indica que el motor tiene que girar para bajar y PWM. INA e INB sólo marcan el sentido de giro, mientras que la señal PWM dice con qué velocidad tiene que girar.

Por lo tanto en la FPGA se implementa la ganancia del regulador y la saturación hasta +-24 voltios que es lo que puede dar la fuente. Como Gpwm hace la conversión de 24 voltios/100 esto supone que a la hora de programar el límite es 100 y no 24.



El proceso de calcular la posición deseada (POS_DESEADA) desde la RAM se explica a continuación.

Utilizando un vector auxiliar llamado SINCRONIZADOR_RAM se une la parte alta y la parte baja del mismo dato, que provienen de la RAM, usando una operación MOD 2. Si el resultado es 1, es decir, es impar el bloque de RAM anterior (-1) que contiene los 5 bits más significativos se introducen en los 5 bits de la parte alta del vector SINCRONIZADOR_RAM, y los del bloque impar en la parte baja. Para ello CNT_READ_RAM tiene que ser mayor que 0.

```
process (data,CNT_READ_RAM,SINCRONIZADOR_RAM,IU_ESTADO)
begin
    nSINCRONIZADOR_RAM<=SINCRONIZADOR_RAM;
    if CNT_READ_RAM>0 then
        if ((CONV_INTEGER (CNT_READ_RAM)) mod 2) = 1 then
            nSINCRONIZADOR_RAM(9 downto 5)<=(SIGNED(ram_block(CONV_INTEGER(CNT_READ_RAM-1))))
            nSINCRONIZADOR_RAM(4 downto 0)<=(SIGNED(ram_block(CONV_INTEGER(CNT_READ_RAM))));
        end if;
    end if;
end process;
```

En la máquina de estados UART_ESTADO se controla el valor de otro vector llamado SALIDA_RAM. Así en el caso del estado STOP, SALIDA_RAM permanece invariable y se consigue la parada del motor. Este vector se actualiza cada 10 ms. que es el tiempo en el que esta troceada la trayectoria a seguir por el motor.

SALIDA_RAM se actualiza con el valor de SINCRONIZADOR_RAM que contiene el dato completo.

```
nSALIDA_RAM<=SINCRONIZADOR_RAM;
```

SALIDA_RAM se actualiza con el valor anterior si se produce un STOP, lo que permite la parada.

```
nSALIDA_RAM<=SALIDA_RAM;
```

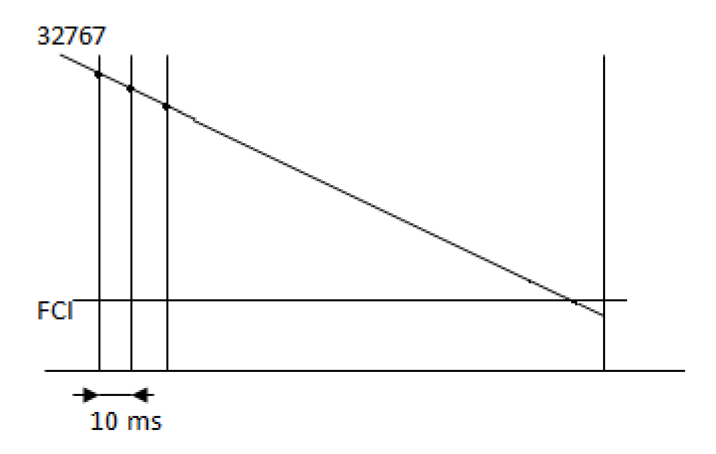
Después de SALIDA_RAM se calcula el vector CONSIGNA que depende de IU_ESTADO.

```
process (SALIDA_RAM,IU_ESTADO,CONSIGNAx256,CONSIGNABAJARx256,CNT13,UART_ESTADO)
begin
nCONSIGNABAJARx256<=CONSIGNABAJARx256;
nCONSIGNAx256<=CONSIGNAx256;
nCNT13<=CNT13;
    if IU_ESTADO=BAJAR_TOPE then
        nCNT13<=CNT13+1;
        if UART_ESTADO=STOP then
        nCONSIGNABAJARx256<=CONSIGNABAJARx256;
        else
        nCONSIGNABAJARx256<=CONSIGNABAJARx256-205;
        end if;
        CONSIGNA<=CONSIGNABAJARx256(21 downto 8);
    elsif IU_ESTADO=ESPERAR then
        nCNT13<=CNT13+1;
        CONSIGNA<=(others=>'0');---POSICION 0
    elsif IU_ESTADO=AJUSTAR then
        if UART_ESTADO=STOP then
        nCONSIGNAx256<=CONSIGNAx256;
        else
        nCONSIGNAx256<=CONSIGNAx256+SALIDA_RAM;
        end if;
        CONSIGNA<=CONSIGNAx256(21 downto 8);
    elsif IU_ESTADO=CONTROL then
        CONSIGNA<=SALIDA_RAM;
    else
        nCONSIGNAx256<=(others=>'0');
        nCONSIGNABAJARx256<="0010011110001000000000";--2530+8ceros=> "111111111
        CONSIGNA<="00100111100010";--2530=>+200=2730*12=32760
        nCNT13<=(others=>'0');
    end if;
end process;
```

Para el estado BAJAR_TOPE, si no hay ningún STOP (en cuyo caso permanece con el mismo valor que en el estado anterior), con un vector auxiliar llamado CONSIGNABAJARx256 se calcula una velocidad fija de bajada de la siguiente manera.

El motor baja a una velocidad constante desde cualquier punto en el que se encuentre hasta el final de carrera inferior. No se sabe la posición en la que se encuentra el motor en ese instante lo que implica que se desconoce el valor del contador del encoder. Por lo tanto, se le asigna un valor al contador del encoder, que podría ser cualquiera, pero como en bucle abierto tiene asignado "111111111111111" (32.767) del proyecto anterior, se aprovecha este valor para cuando se haga la transición de bucle abierto a bucle cerrado no sea un problema.

La trayectoria que sigue para bajar es una recta representada en la siguiente figura.



Donde se calcula la cada 10 ms. el nuevo punto. Por criterio propio se eligió una velocidad constante de 8 mm/s que como máximo tardará 12 segundos y medio en bajar desde la posición más elevada que es de 100 mm. o sea los 10 cm que mide el vástago.

$$\frac{8mm}{s} = \frac{0{,}08mm}{10ms}$$

Para no trabajar con decimales se añaden 8 bits a la derecha en el vector, de ahí el nombre CONSIGNABAJARx256 porque añadir 8 bits a la derecha es lo mismo que multiplicar por 256. 0,08mm. equivalen a 0,8 décimas de milímetro, que es la precisión con la que se trabaja.

0,8 décimas de milímetros multiplicadas por 256 son 204,8. Como no se trabaja con decimales se redondea a 205. Cuya velocidad será:

$$\frac{205}{256} = \frac{0{,}80078125 \ décimas \ de \ milímetro}{10ms}$$

$$8{,}0078125 \frac{mm}{s}$$

Esto lo único que supone es, que en vez de en 12 segundos y medio que tarda el peor caso en bajar desde arriba del todo hasta abajo, es decir, en recorrer los 100 mm. que mide el vástago, tardará:

$$\frac{100mm}{\frac{8{,}0078125mm}{s}} = 12{,}48\ segundos$$

Lo que supone un error muy pequeño.

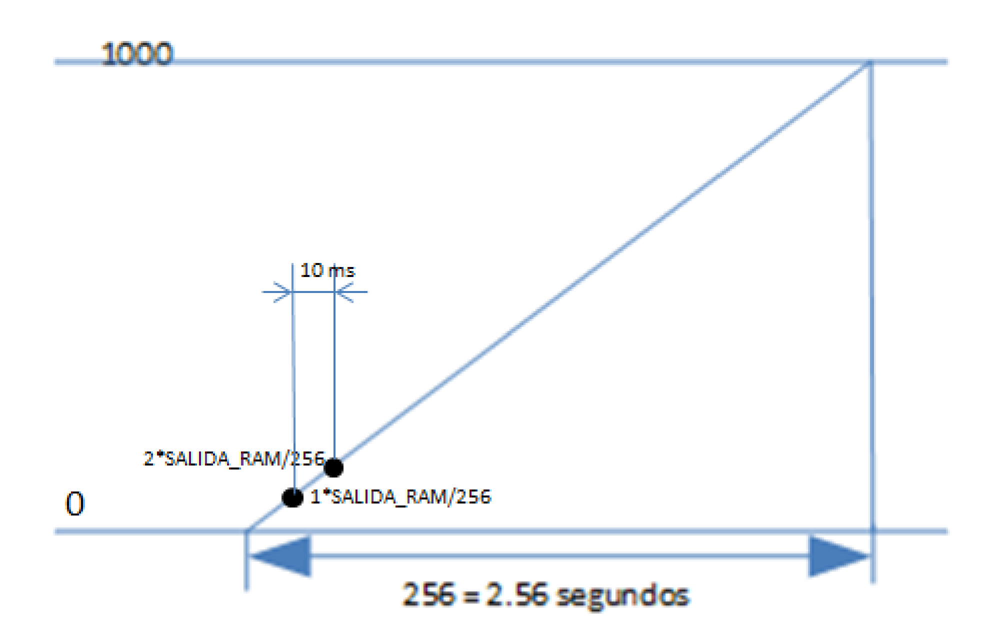
Como previamente se ha multiplicado por 256, a la hora de dar valor a CONSIGNA los 8 bits menos significativos de CONSIGNABAJARx256 no se tienen en cuenta.

CONSIGNABAJARx256 se inicializa con el valor "10011110001000000000". Este valor es debido a que para que equivalga a "111111111111" que vale el contador encoder. Como CONSIGNA antes de llegar a POS_DESEADA se le suma 200 y se multiplica por 12, deshaciendo el camino, "111111111111"=32.767 que dividido entre 12 es 2730, menos 200 es 2530, que en binario es "100111100010", que con 8 bits que se añaden a la derecha es "10011110001000000000", es decir que para que POS_DESEADA sea todo 1 y por lo tanto igual a CNT_ENCODER y así que no se produzca ninguna diferencia, CONSIGNABAJARx256 tiene que ser "10011110001000000000". Aun así se produce un ligero error debido a que (2530+200)*12 es 32.760 ("111111111111000") y no 32.767 ("111111111111111") que es valor del que se partía.

El contador llamado CNT13 se utiliza para llevar la cuenta de los segundos que le cuesta bajar al motor. Como se ha mencionado antes, el caso en el que tarda más en bajar es cuando tiene que recorrer todo el vástago, con 12,48 segundos. Como en este proyecto la idea es de controlar no sólo un motor si no más de uno, el primero que llega al final de carrera inferior de su vástago tiene que esperar al resto. Por lo tanto, la condición para que se inicie el nuevo estado AJUSTAR es que el contador CNT13 sea " 10100010100" (1300) por que tiene un reloj de 10 ms. lo que equivale a 13 segundos. Tras este tiempo todos los motores habrán llegado abajo. A CONSIGNA se le asigna el valor de 0 en el estado ESPERAR como referencia.

El estado AJUSTAR es parecido a BAJAR_TOPE. De la RAM se extrae la primera posición a la que tiene que llegar el motor. Se utiliza un vector auxiliar llamado CONSIGNAx256 que de la misma manera que CONSIGNABAJARx256 se le ha añadido 8 bits más. CONSIGNAx256 empieza con el valor 0 y se le suma la primera posición de la memoria RAM, es decir se le añade SALIDA_RAM. Como luego los 8 bits menos significativos se desprecian, la consecuencia es que el incremento es de SALIDA_RAM/256. Como este vector tiene un reloj de 10ms. tras 2,56 segundos (o 256 iteraciones) se llega a la primera posición.

$$\frac{SALIDA_RAM}{256} * 256 = SALIDA_RAM$$



En el estado CONTROL CONSIGNA es directamente SALIDA_RAM.

CONSIGNA contiene el valor deseado en décimas de milímetro, mientras que CNT_ENCODER se mide en pulsos del encoder. Antes de poder compararlos y sacar el error es necesario que CONSIGNA tenga las mismas unidades. Para ello se le suma 200 y se multiplica por 12.

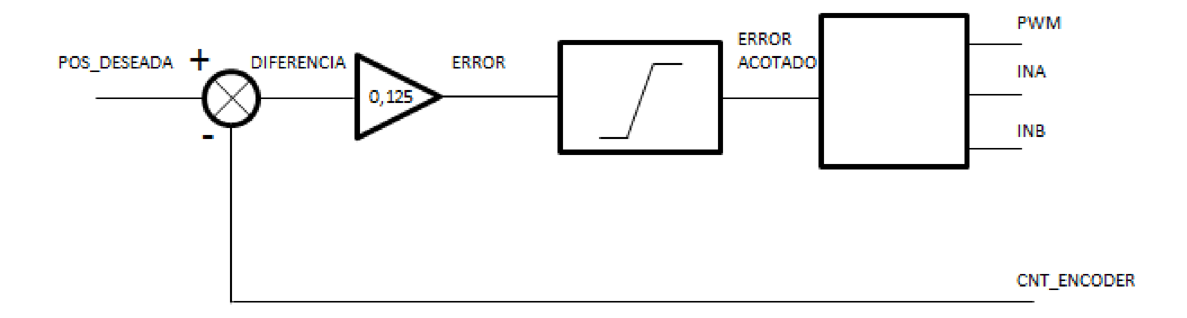
Se le suma 200 para que no se desborde el contador encoder. Si éste fuera 0 y por la inercia del motor en movimiento se le restase 1 al vector, esto implicaría que su valor sería de todo unos e inmediatamente con el control en bucle cerrado el motor intentaría alcanzar esta posición, causando un pico de corriente.

Se multiplica por 12 porque una décima de milímetro equivale a 12 pulsos del encoder. Para ello se multiplica por 16 y por 4 mediante dos vectores auxiliares y se realiza la resta. Así las operaciones son sencillas, con dos desplazamientos hacia la izquierda para realizar la multiplicación y una resta.

$$12xConsigna = 16xConsigna - 4xConsigna$$

Con todo esto se tiene la posición deseada lista para ser comparada con el contador encoder.

Sólo queda implementar el bucle cerrado.



Para calcular el regulador proporcional se han utilizado valores múltiplos de dos para que la multiplicación por la constante sea un desplazamiento del registro hacia la izquierda o la derecha. En este caso, para el motor en vacío, la constante del regulador es de 0,5, lo que equivale a dividir entre 2. Para el motor en su estructura es de 0,125 o lo que es lo mismo, dividir entre 8.

Cuando se calcula el error, se tiene en cuenta la extensión de signo porque éste puede ser positivo o negativo.

Error acotado limita el error a +-100, lo que equivale a +-24 en la fuente de alimentación.

```
process(CNT_PWM)
begin
  if (CNT_PWM=90) then
    nCNT_PWM <= (others=>'0');
  else
    nCNT_PWM <= CNT_PWM + 1;
  end if;
end process;
```

Para la salida PWM se compara el valor absoluto (porque el signo del error se usa sólo para el sentido del giro, no para esto) con el contador CNT_PWM de frecuencia 0,1ms. que se usaba en bucle abierto.

Para el sentido de giro, si el error es positivo es porque el motor tiene que subir. Si el error es negativo tiene que bajar, y si no es ni positivo ni negativo, no tiene nada que hacer.

```
if ERROR_ACOTADO>0 then
    INA<='1';
    INB<='0';
elsif ERROR_ACOTADO<0 then
    INA<='0';
    INB<='1';
else
    INA<='0';
    INB<='0';
end if;
```

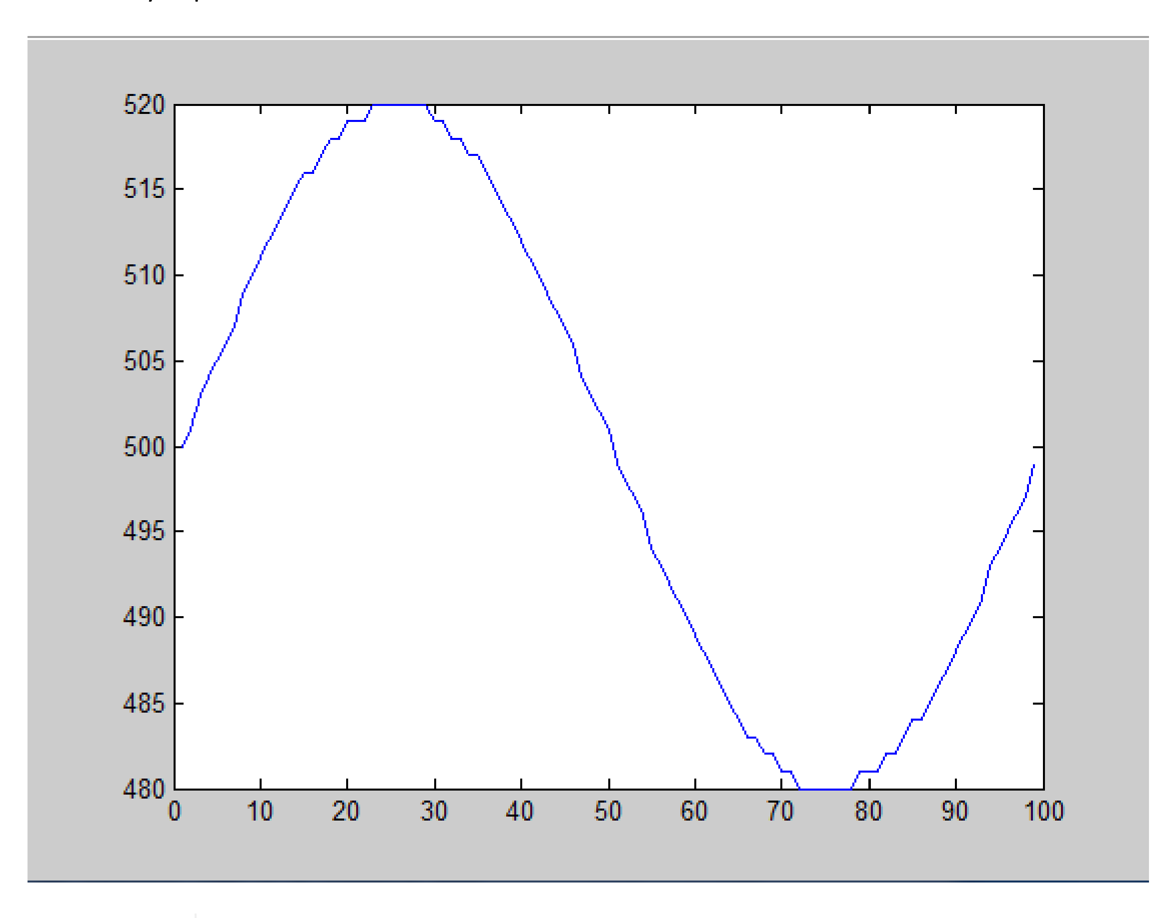
Las salidas de sentido de giro sólo se producen en algún estado de movimiento del motor, es decir, en el estado BAJAR_TOPE, AJUSTAR y CONTROL.

CAPÍTULO 4

4 Programación en MATLAB para el envío y recepción de datos desde el ordenador a la FPGA.

Se han programado dos trayectorias para el motor. La primera es una senoide, para simular un movimiento de vaivén suave. La segunda es un escalón con el que se comprueba el tiempo de respuesta y la posibilidad o no de sobreoscilación.

Para la senoide los valores que se introducen son la amplitud, el número de divisiones y el punto de inicio. Mientras que para el escalón son la altura del escalón, el número de divisiones y el punto de inicio.



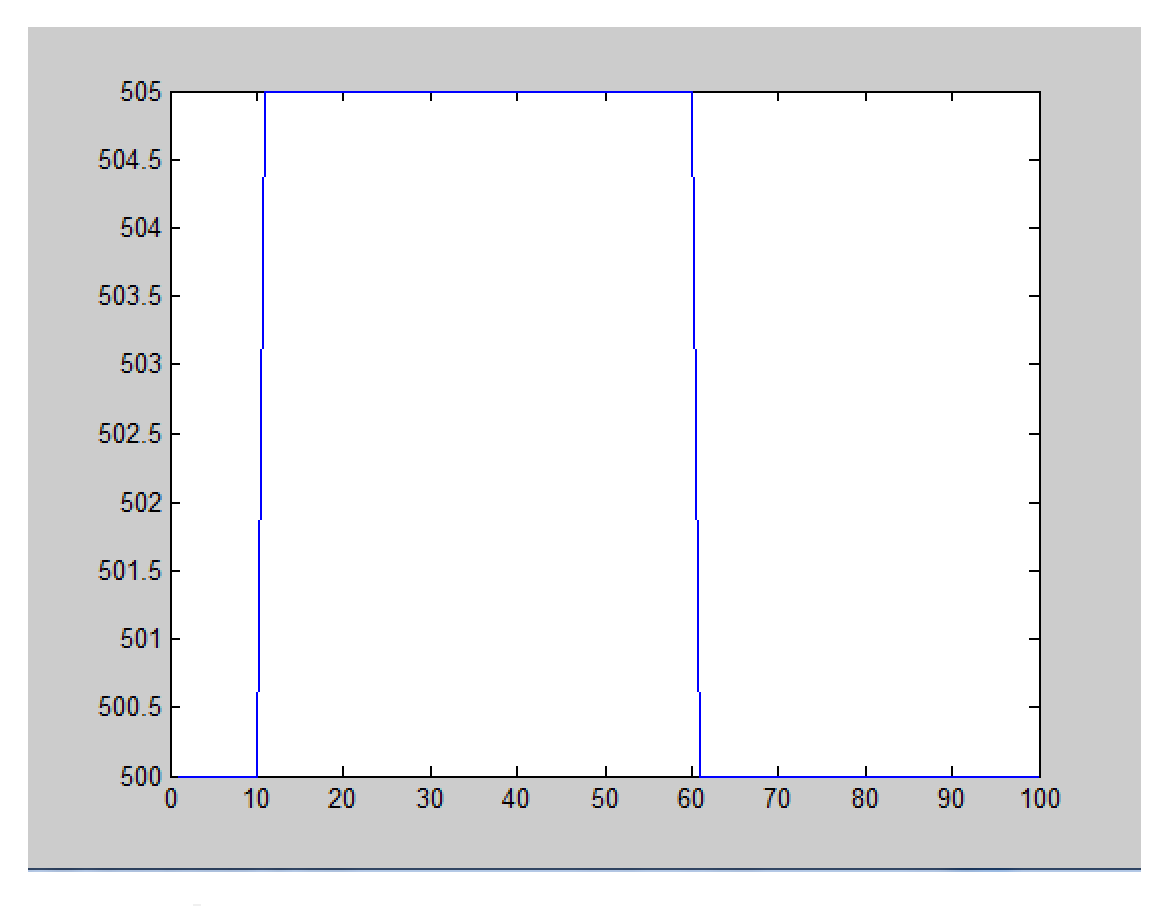

```
ingrese el valor de amplitud de la senoide = 20
ingrese el número de divisiones = 100
ingrese el punto de inicio (de 0 a 1000)= 500
```

El punto de inicio indica la altura a la que empieza el movimiento senoidal, en este caso 500 décimas de milímetro, que son 5 cm.

El número de divisiones indica indirectamente el periodo de la senoide. Cada división es un dato que se envía a la FPGA y cada 10 milisegundos sale hacia el motor. Por lo que el periodo es de:

$$10\ ms.x Número\ de\ divisiones$$


Así pues, en el ejemplo, el periodo será de 100*10 ms. que equivale a 1 segundo.




```
ingrese el valor de la altura del escalón = 5
ingrese el número de divisiones = 100
ingrese el punto de inicio = 500
```

En el caso del escalón, el punto de inicio sirve para lo mismo que para el de la senoide.

El número de divisiones permite darle tiempo entre flancos de subida y bajada. La subida se produce, por criterio propio, cuando ha transcurrido Número de divisiones/10 y la bajada cuando ha pasado Número de divisiones*6/10. Así el tiempo en el que está arriba y el que está abajo es el mismo.

El máximo recorrido que tiene el motor viene delimitado por los finales de carrera superior e inferior y es de 1000 décimas de milímetro. Con el programa MATLAB no se ha tenido en cuenta este impedimento, por lo que puede darse que con la suficiente amplitud de la senoide o altura del escalón, sumado a la altura del punto de inicio se activen los finales de carrera. Queda en manos del usuario la elección de los valores que no sobrepasen los 10 cm.

Al correr el programa, se obtienen dos archivos, uno llamado archivoconsigna.txt, que contiene la trayectoria enviada y otro llamado archivoencoder.txt, que contiene la trayectoria real medida por el encoder.

4.1 Senoide.

Se define el puerto de escritura/lectura PS:

PS=serial('COM4');

set(PS,'Baudrate',57600); % se configura la velocidad a 57600 Baudios

set(PS,'StopBits',1); % se configura bit de parada a uno

set(PS,'DataBits',8); % se configura que el dato es de 8 bits

set(PS,'Parity','even'); % se configura sin paridad

set(PS,'FlowControl','none');%Sin control de hardware

Con una variable llamada estado se controla todos los estados de UART_ESTADO.

estado=input('INTRODUZCA EL NÚMERO.- 1 stop,2 start,3 mandar_posicion_on,4 mandar_posicion_off,5 volver_a_mandar,6 salir,7 ver posición = ');

Al principio del programa se le asigna un 8 a esta variable y con un bucle while con la condición de que si estado no es 6 se permanezca en el bucle (condición de salida).

Para generar la senoide se pide la amplitud de ésta, el número de divisiones, que indirectamente es el período de ésta. Cada división equivale a un dato cada 10 ms., por lo que 50 divisiones por ejemplo equivalen a 0,5 segundos. El punto de inicio equivale a la altura donde empieza el movimiento de la senoide, la posición de 0 a 10 cm. del vástago.

```
a=input('ingrese el valor de amplitud de la senoide = '); %amplitud
d=input('ingrese el número de divisiones = '); %divisiones
puntoinicio=input('ingrese el punto de inicio (de 0 a 1000)= '); %puntoinicio
x=linspace(0,2*pi,d);
y=a*sin(x)+puntoinicio;
```

Con linspace se crea una recta de 0 a 2pi con d divisiones para x y con sin, una senoide para y.

```
j=round(y(i));
q(i)=j;
```

Se redondea el valor de y. En q se guarda todos los valores de j, que son los que se envían a la FPGA.

binario=fi(j,0,10,0);

El valor de j se transforma a binario con 10 bits, para ser dividido en 2 vectores de 5 bits.

```
if i==1
        partebajabyte=bitget(binario,[5:-1:1]);
        %disp(bin(partebajabyte))
        partealtabyte=bitget(binario,[10:-1:6]);
        %disp(bin(partealtabyte))
        enviar=32+bin2dec(bin(partealtabyte));%+32=> añadimos '001' como bits más
significativos
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        enviar=bin2dec(bin(partebajabyte));
        fwrite(PS,enviar);
end
```

Si es el primer dato a enviar se le suma 32 que equivale a sumar “00100000” con lo que UART_ESTADO entiende que es el primer dato. Si no, se envía el resto.

```
if i>2
        partebajabyte=bitget(binario,[5:-1:1]);
        %disp(bin(partebajabyte))
        partealtabyte=bitget(binario,[10:-1:6]);
        %disp(bin(partealtabyte))
        enviar=bin2dec(bin(partealtabyte));
        fwrite(PS,enviar);
        enviar=bin2dec(bin(partebajabyte));
        fwrite(PS,enviar);
end

fid=fopen('archivoconsigna.txt','w');
fprintf(fid,'%i \n',q);
load archivoconsigna.txt
figure
plot(archivoconsigna)
fclose(fid);
```

En el archivoconsigna.txt se guarda los valores enviados a la FPGA contenidos en q, y se dibuja.

```
enviar=64;
fwrite(PS,enviar);
fclose(PS);
```

Si no es el primer dato se envía sin sumar nada. Cuando se acaban los datos se suma 64 que equivale a “1000000” que se interpreta como último dato.

```
elseif estado==1
        enviar=128;
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        fclose(PS);
elseif estado==2
        enviar=96;
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        enviar=0;
        fwrite(PS,enviar);
        fclose(PS);
elseif estado==3
        enviar=160;
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        fclose(PS);
elseif estado==4
        enviar=192;
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        fclose(PS);
if estado==5
        enviar=224;%ASI TERMINA EL ESTADO EN EL QUE ESTE Y EMPIEZA EN RESET
        fopen(PS); %Abre objeto
        fwrite(PS,enviar);
        fclose(PS);
        estado=8;
end
```

Estado=1 equivale a STOP y se suma 128 que es “10000000”.

Estado=2 equivale a START y se suma 96 que es “01100000”, se envía otro dato adicional porque DRX guardará el valor de “01100000” y si se produce un STOP por los finales de carrera, como se queda guardado el START, no parará. Si no se envía se saldrá del vástago.

Estado=3 equivale a MANDAR_POS_ON y se suma 160 que es “10100000”.

Estado=4 equivale a MANDAR_POS_OFF y se suma 192 que es “11000000”.

Estado=5 inicializa las variables de la FPGA como son contador encoder y las consignas de bajar tope y de ajustar para que no se produzcan errores.

```
fopen(PS);
sizebuffer=6;
l=0;
n=0;
datosaleer=input('ingrese el número de datos que quiere leer = ');
intnumerocompleto=2;
while intnumerocompleto~=1
        datos= fread(PS,sizebuffer,'uchar');
        binary=fi(datos,0,8,0);
        for h=1:sizebuffer
                if mod(h,6)==3
                        if mod(h,2)==1
                                numerocompleto=bitconcat(binary(h),binary(h+1));
                                %disp(bin(numerocompleto))
                                intnumerocompleto=bin2dec(bin(numerocompleto));
                        end
                end
        end
end
```

Para leer los datos de la FPGA, desde ésta se envía de la siguiente manera:

```
-- Registro de datos a enviar por la UART
nREG_DTX(0) <= "00"&nCNT_ENC(13 downto 8);
nREG_DTX(1) <= nCNT_ENC(7 downto 0);
nREG_DTX(2) <= "0000"&nCNT_READ_RAM(11 downto 8);
nREG_DTX(3) <= nCNT_READ_RAM(7 downto 0);
nREG_DTX(4) <= "0000000"&NLD7;
nREG_DTX(5) <= "00000000";
```

Se leen de 6 en 6 datos, de la misma manera que se manda desde la FPGA de 6 en 6.

Con datosaleer se indica cuantos datos se quieren leer.

Haciendo la operación mod(h,6)=3 se lee nREG_DTX(2) y se concatena con el siguiente vector nREG_DTX(3). Intnumerocompleto equivale al CNT_READ_RAM, si éste es 1 es porque se inicializa la lectura de la RAM y por lo tanto a partir de aquí se leerá el CNT_ENC empezando a describir la senoide.

```
for l=1:datosaleer
        for h=1:sizebuffer
                if mod(h,6)==1
                        if mod(h,2)==1
                        numerocompleto=bitconcat(binary(h),binary(h+1));
                        %disp(bin(numerocompleto))
                        intnumerocompleto=bin2dec(bin(numerocompleto));
                        posreal=intnumerocompleto/12-200;
                        z(l)=posreal;
                        end
                end
        end
        datos= fread(PS,sizebuffer,'uchar');
        binary=fi(datos,0,8,0);
end
fid=fopen('archivoencoder.txt','w');
fprintf(fid,'%i \n',z);
load archivoencoder.txt
figure
plot(archivoencoder)
fclose(fid);
clear z;
fclose(PS);
```

A partir de la inicialización del contador de la lectura de la RAM se almacenan los datos recibidos del encoder en archivoencoder.txt en décimas de milímetro.

Por lo que al final se tiene dos archivos, uno llamado archivoconsigna.txt que contiene los datos en décimas de milímetro que se envían, y archivoencoder.txt que contiene los datos en décimas de milímetro que se reciben en sincronismo con la consigna.

4.2 Escalón.

```
al=input('ingrese el valor de la altura del escalón = '); %altura
d=input('ingrese el número de divisiones = '); %divisiones
puntoinicio=input('ingrese el punto de inicio = '); %punto de inicio
referencia=puntoinicio;
p1=round(d/10);
p2=round(6*d/10);
```

Para el escalón se introduce la altura del escalón, el número de divisiones que como la senoide, cada división equivale a 10 ms, el punto de inicio (de 0 a 1000 décimas de milímetro) y se calculan dos puntos adicionales. P1 para el flanco de subida del escalón y p2 para el flanco de bajada.

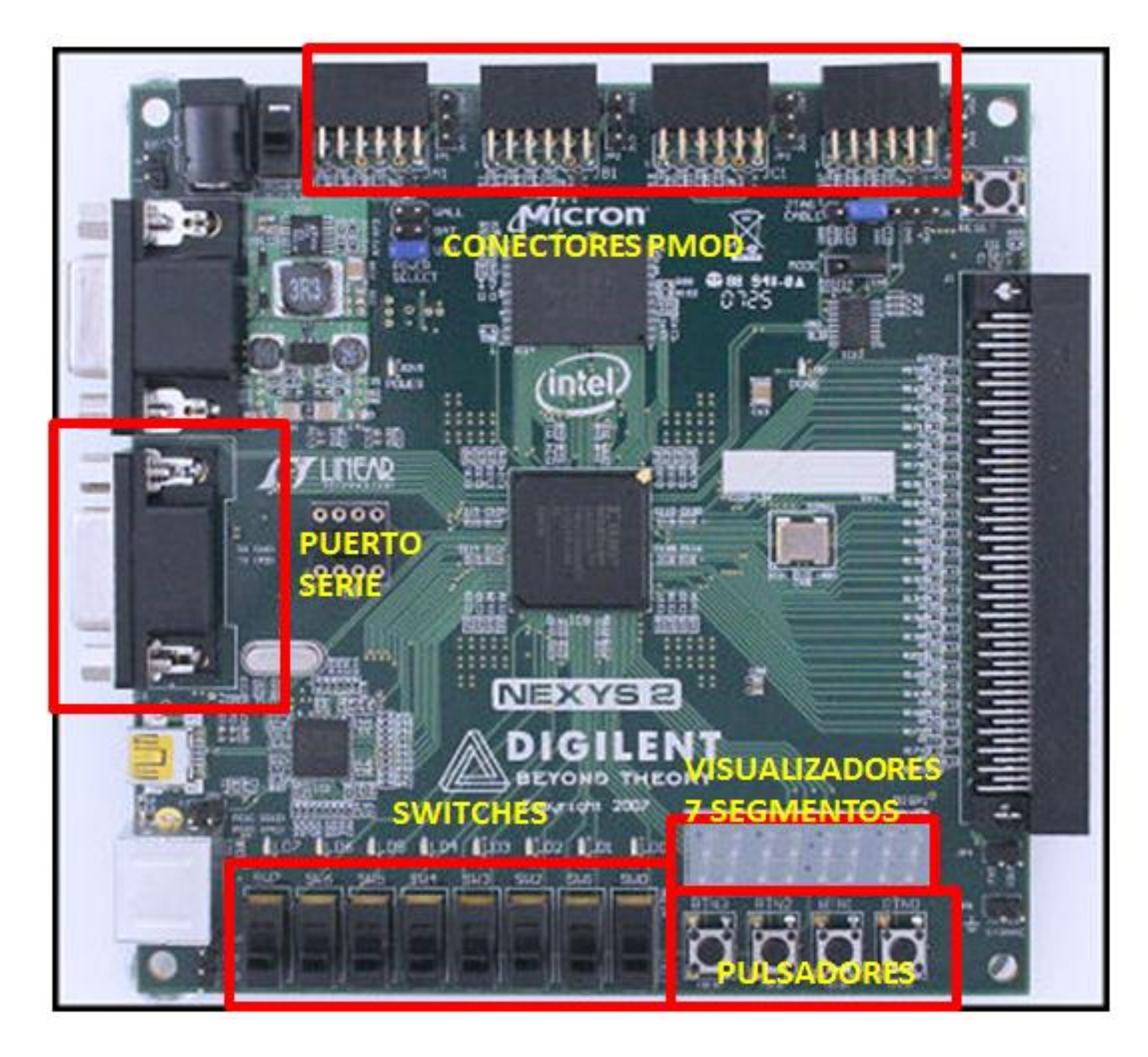
Para leer datos, como en la senoide, simplemente se lee el contador encoder desde el momento en el que CNT_READ_RAM es 1 (se elige 1 y no 0 porque el CNT_READ_RAM se actualiza cada 5 ms. mientras que los datos se envían cada 10 ms. por lo que CNT_READ_RAM siempre es impar al ser enviado).

CAPÍTULO 5

5 Conexión del actuador.

5.1 FPGA utilizada.

La placa con la que se ha trabajado es una Nexys2 1200 de DIGILENT.



A través del puerto serie se reciben los datos del ordenador. En los conectores PMOD van las placas de potencia. Los switches y pulsadores se explican a continuación.

5.2 Interfaz de usuario de la FPGA.



SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0		BTN3	BTN2	BTN1	BTN0
ILIM Disable	FC Disable		BUCLE	CICLO	STEP	BAJA	SUBE		RST		DN	UP

ILIM Disable desactiva la limitación de instensidad.

FC Disable desactiva los finales de carrera, ¡muy peligroso para el actuador! Se encienden los leds LD1 y LD0.

BUCLE, para elegir control en bucle abierto ‘0’ o bucle cerrado ‘1’.

Cuando se trabaja en bucle cerrado no se utilizan los demás interruptores y pulsadores de la placa. Para trabajar en bucle abierto se utilizan los siguientes:

- UP y DN sube y baja la referencia del PWM (0-9) que se visualiza en 7 segmentos, excepto cuando se activa STEP.
- SUBE hace subir el actuador hasta que se desactiva o se alcanza FCS.
- BAJA hace bajar el actuador hasta que se desactiva o se alcanza FCI.
- STEP realiza una subida o bajada de 10 mm cada vez que se pulsa UP o DN.
- CICLO repite continuamente el consistente en subir 20 mm y luego bajar 20 mm.

El circuito lleva una UART que envía una trama de datos del actuador cada periodo de muestreo TS=10 ms. La trama de 6 datos DTX(0 to 5) consiste en:

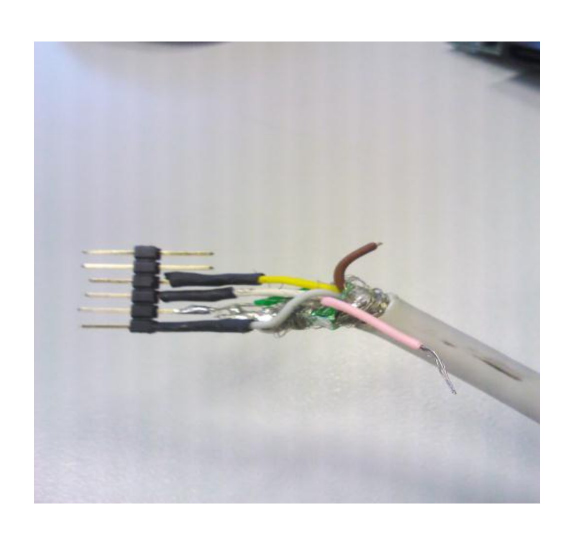
- DTX(0): Byte alto del contador del encoder.
- DTX(1): Byte bajo del contador del encoder.
- DTX(2): Byte alto del contador de lectura de la RAM.
- DTX(3): Byte bajo del contador de lectura de la RAM.
- DTX(4): Registro de pruebas sin uso.
- DTX(5): Registro de pruebas sin uso.

El contador del encoder es usado para verificar que el motor sigue la trayectoria enviada.

El contador de lectura de la RAM se usa para saber cuándo empieza la lectura de la trayectoria almacenada en la memoria RAM.

5.3 Conexión del encoder.

El encoder montado inicialmente es un HEDS-5645-G13 de 360 pulsos por vuelta. Tiene montado un cable con los colores de la tabla. Requiere alimentación entre 4.5 y 5.5 V y resistencias de pull-up de 2.7k en CHA y CHB.



amarillo	CHB
blanco	CHA
verde	GND
gris	VCC

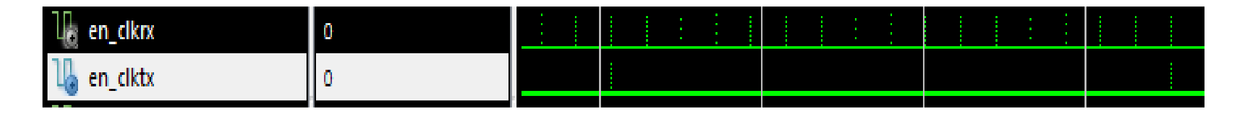
5.4 Finales de carreras FCI y FCS.

Constituidos por sensores Hall A3214 que se alimentan entre 2.5 y 3.5 V. Son omnidireccionales, con salida activa en bajo. El tiempo de respuesta va entre 60 y 90 ms porque están dormidos la mayoría del tiempo (dc = 0.1%). Sus conectores van al conector de arriba de los conectores JC (FCS) y JD (FCI).

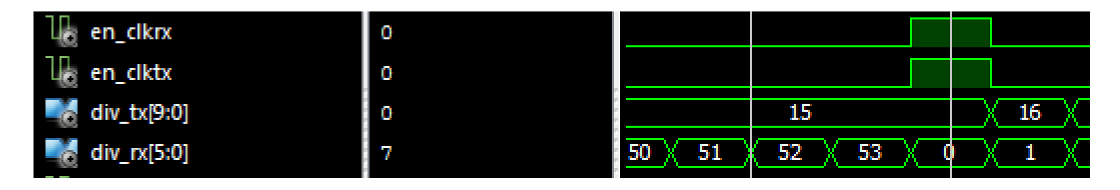
Cambiado el FCI por un A3245 cuyo rango de alimentación va de 3,6 a 24 V. Ojo cambiar el jumper del PMOD de la FPGA para alimentarlo a 5V.

CAPÍTULO 6

6 Simulaciones.



En esta imagen se observa que por cada enable de la UART para enviar dato (en_clktx) se producen 16 pulsos del enable del receptor (en_clkrx) lo que significa que es 16 veces más rápido este último, debido a que así se ha elegido.



Como se describe en el programa, cuando DIV_RX es 0, EN_CLKRx se activa y cuando este se activa y DIV_TX es 15 EN_CLKTx se activa.

```
process (DIV_RX)
begin
  if (DIV_RX = 53) then
    nDIV_RX <= (others=>'0');
  else
    nDIV_RX <= DIV_RX + 1;
  end if;
end process;

EN_CLKRx <= '1' when (DIV_RX=0) else '0';

-- Clock enable TX (16 veces mas lento que el de RX)

process (DIV_TX,EN_CLKRx)
begin
  if (DIV_TX = 16) then
    nDIV_TX <= (others=>'0');
  elsif EN_CLKRx='1' then
    nDIV_TX <= DIV_TX + 1;
  else nDIV_TX<=DIV_TX;
  end if;
end process;

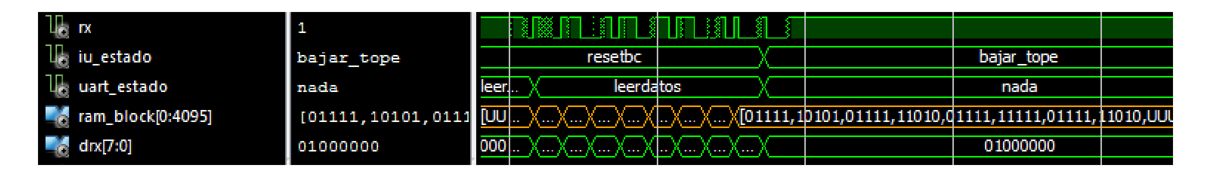
EN_CLKTx <= '1' when (DIV_TX=15) and (EN_CLKRx='1') else '0';
```



Cuando DRX recibe 001 en los 3 bits más significativos, se empiezan a guardar los 5 bits menos significativos de los datos recibidos. Mientras CNT_WRITE_RAM aumenta por cada dato guardado y CNT_READ_RAM permanece invariable.



Finalmente cuando los 3 bits más significativos de DRX son 010 se termina la recepción de datos.



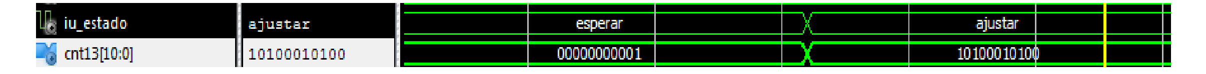
UART_ESTADO comienza en el estado LEERUNAVEZ y cuando RX recibe los datos a través de DRX se almacenan en la RAM. En este caso IU_ESTADO es RESETBC y no cambia a BAJAR_TOPE hasta que no se ha terminado de leer hasta el último dato. Como UART_ESTADO no tiene nada que hacer permanece en el estado NADA, hasta que reciba una orden.



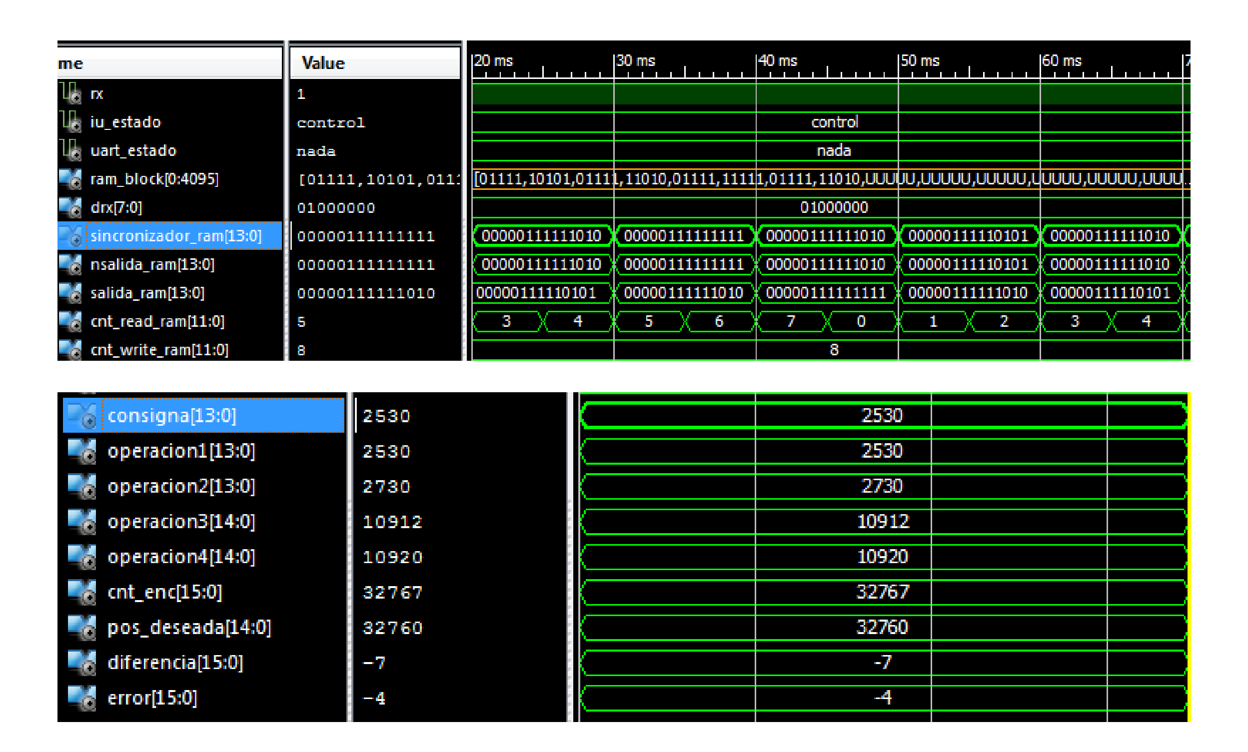
En el estado BAJAR_TOPE cada 10 ms CONSIGNABAJARx256 se reduce en 205 como se ha explicado anteriormente.

```
if IU_ESTADO=BAJAR_TOPE then
    nCNT13<=CNT13+1;
    if UART_ESTADO=STOP then
    nCONSIGNABAJARx256<=CONSIGNABAJARx256;
    else
    nCONSIGNABAJARx256<=CONSIGNABAJARx256-205;
    end if;
```

Cuando se activa el final de carrera inferior el siguiente estado es ESPERAR donde tras 13 segundos seguirá con el estado AJUSTAR. A CONSIGNA se le asigna el valor de referencia 0.

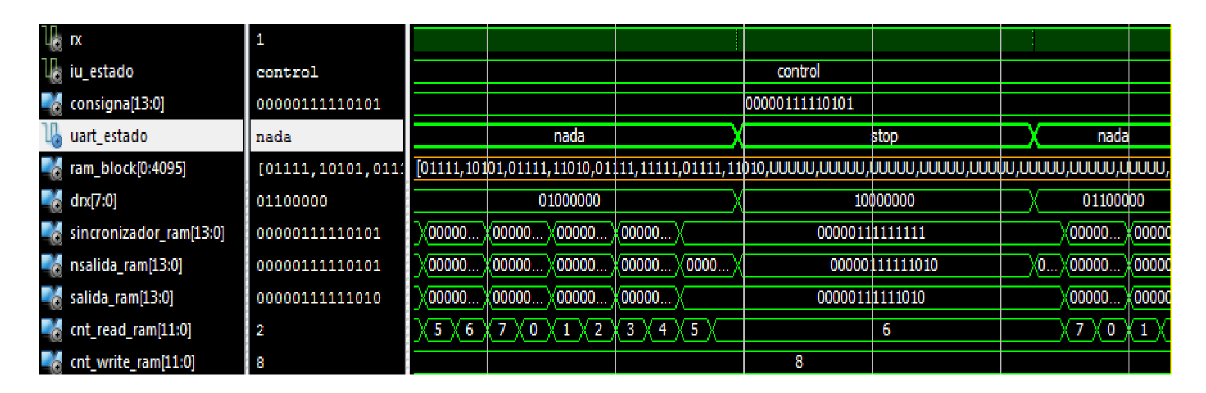


En el estado CONTROL sincronizador_ram recoge el dato de la memoria RAM cada vez que CNT_READ_RAM es impar a la frecuencia de 50 MHz por lo que el cambio se produce al instante, mientras que SALIDA_RAM se actualiza cada 10 ms.



POS_DESEADA dependerá del valor de CONSIGNA, en este caso valdrá 32760 por que CONSIGNA es 2530, que tras pasar por la suma de +200 y la multiplicación por 12 da los 32760 de POS_DESEADA. Si por ejemplo hubiésemos tomado como consigna 2531 saldría 32772 y se desbordaría. Por este motivo desde el principio se produce una diferencia de -7 (32760-32767) cuyo error (diferencia tras ser dividida entre 2) es la mitad, en este caso -4 debido al redondeo.

Como el error es inferior a 100 y superior a -100 no se satura.



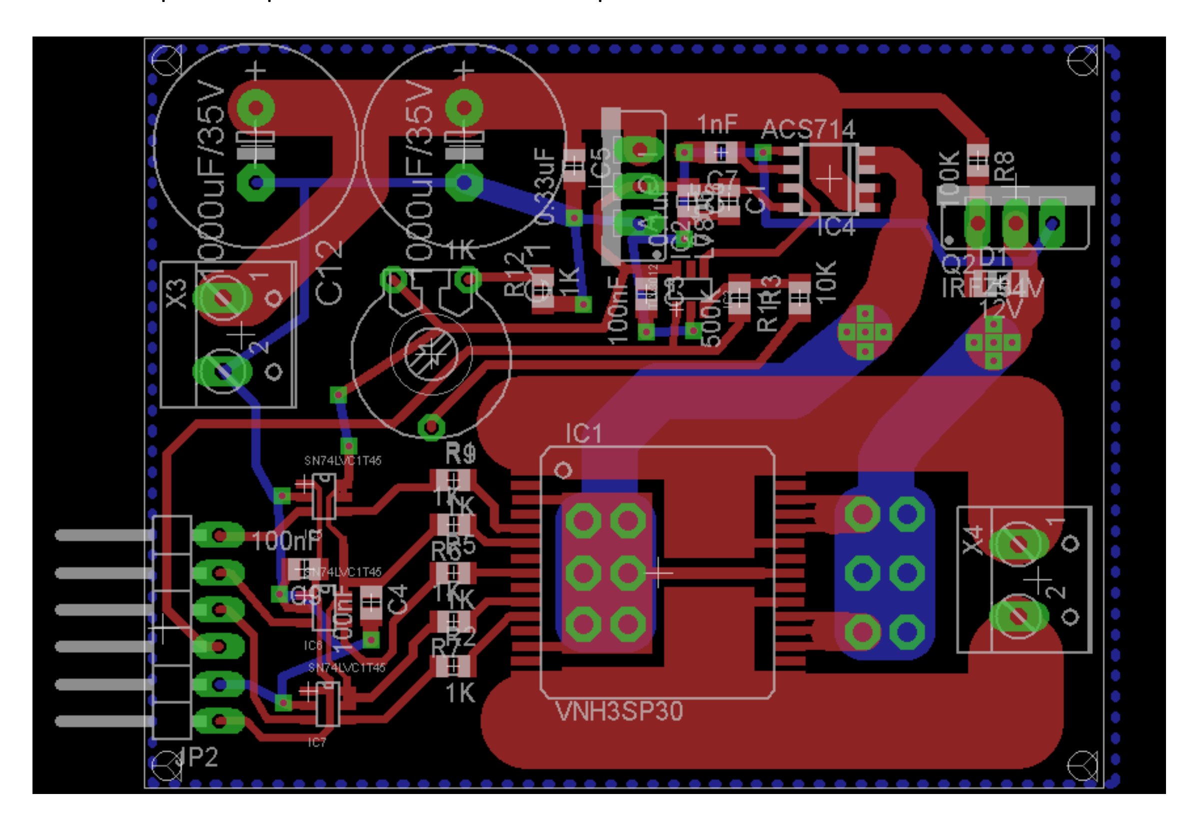
Cuando los 3 bits más significativos de DRX son 100 se produce el estado STOP en UART_ESTADO donde la SALIDA_RAM y CNT_READ_RAM mantienen los valores hasta que se recibe 011 en los 3 bits superiores de DRX, momento en el cual siguen actualizándose los valores porque sigue el movimiento del motor.

CAPÍTULO 7

7 PCB

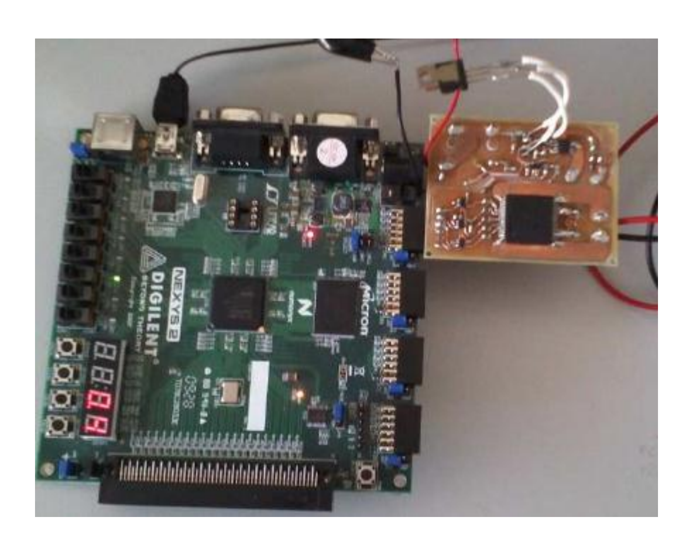
7.1 Reducción de tamaño

La placa de potencia se había diseñado para un solo motor controlado con la FPGA.



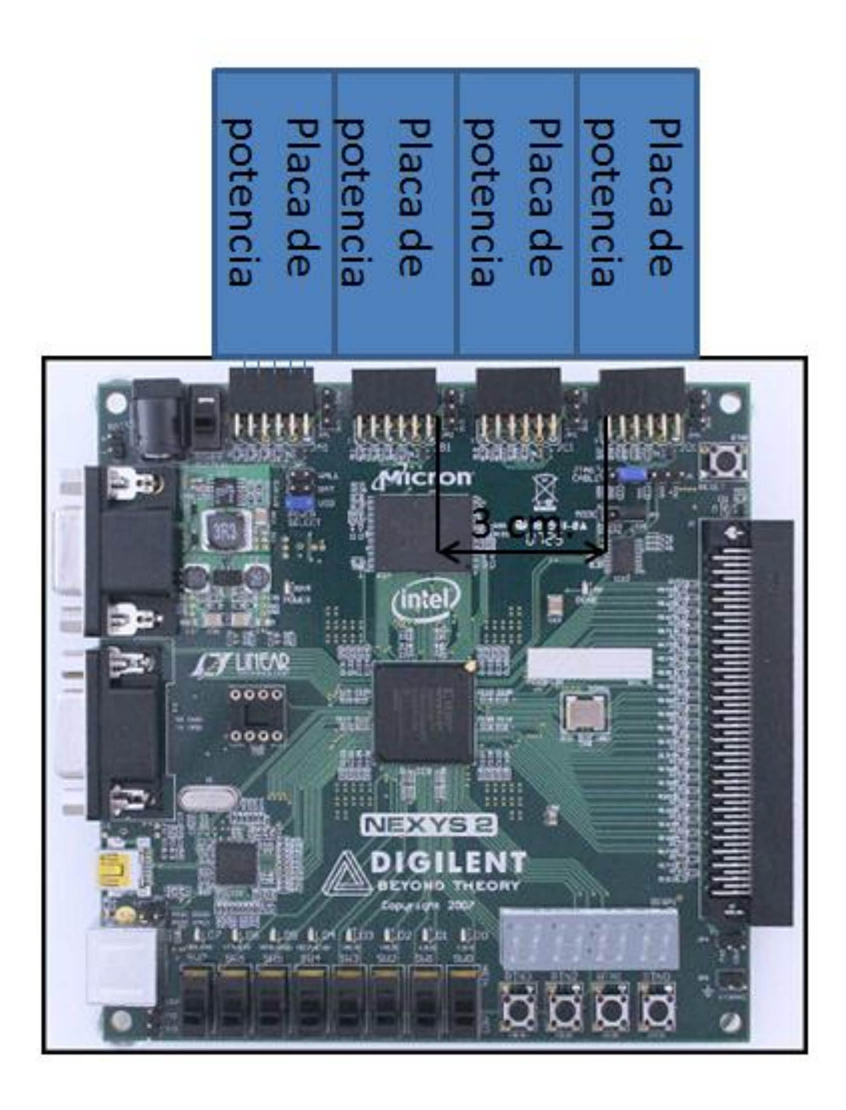
PCB diseño del proyecto anterior.

No se había tenido en cuenta la posibilidad de conectar más de una placa de potencia a la FPGA.

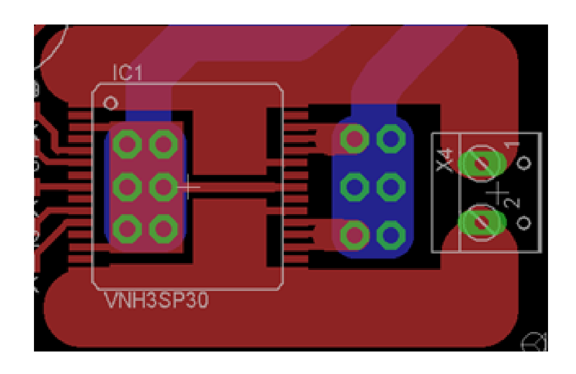


Conexión de la placa de potencia a la FPGA.

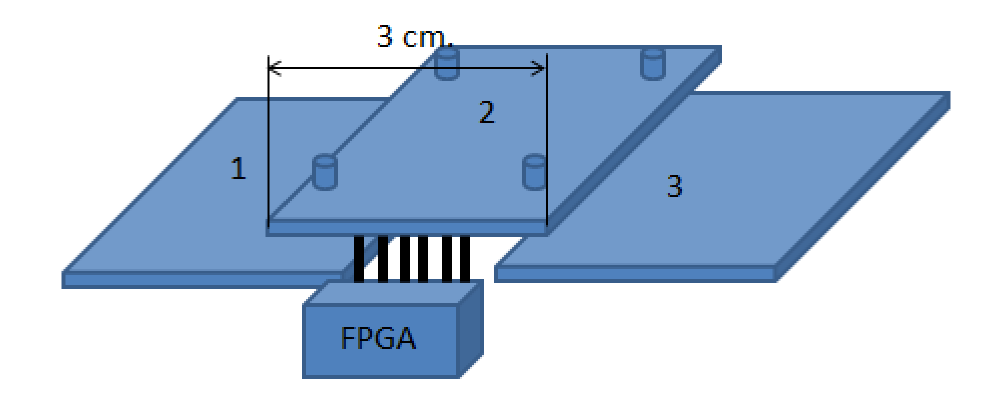
En este proyecto el objetivo es minimizar el tamaño de la placa, para que se puedan aprovechar los otros 3 conectores libres que se ven en la imagen. Con esta FPGA sólo se podrían manejar 4 motores utilizando esos 4 conectores. Para utilizar los 6 motores habrá que elegir una placa con más.

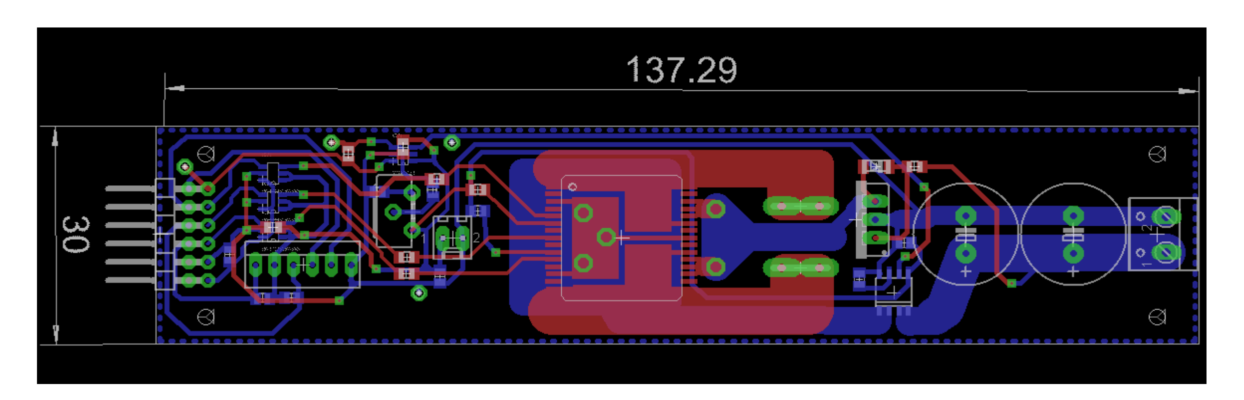


La anchura máxima de las placas de potencia es de 3 cm. como se indica en la imagen. Y la anchura mínima tiene que ser mayor que el ancho de este componente y de sus pistas que disipan calor y que mide poco menos de 3 cm.



Por lo que, a la hora de resolverlo, se ha optado por una disposición como se indica a continuación. Donde con 4 tornillos queda una estructura más compacta y se resuelve el problema de tamaño.



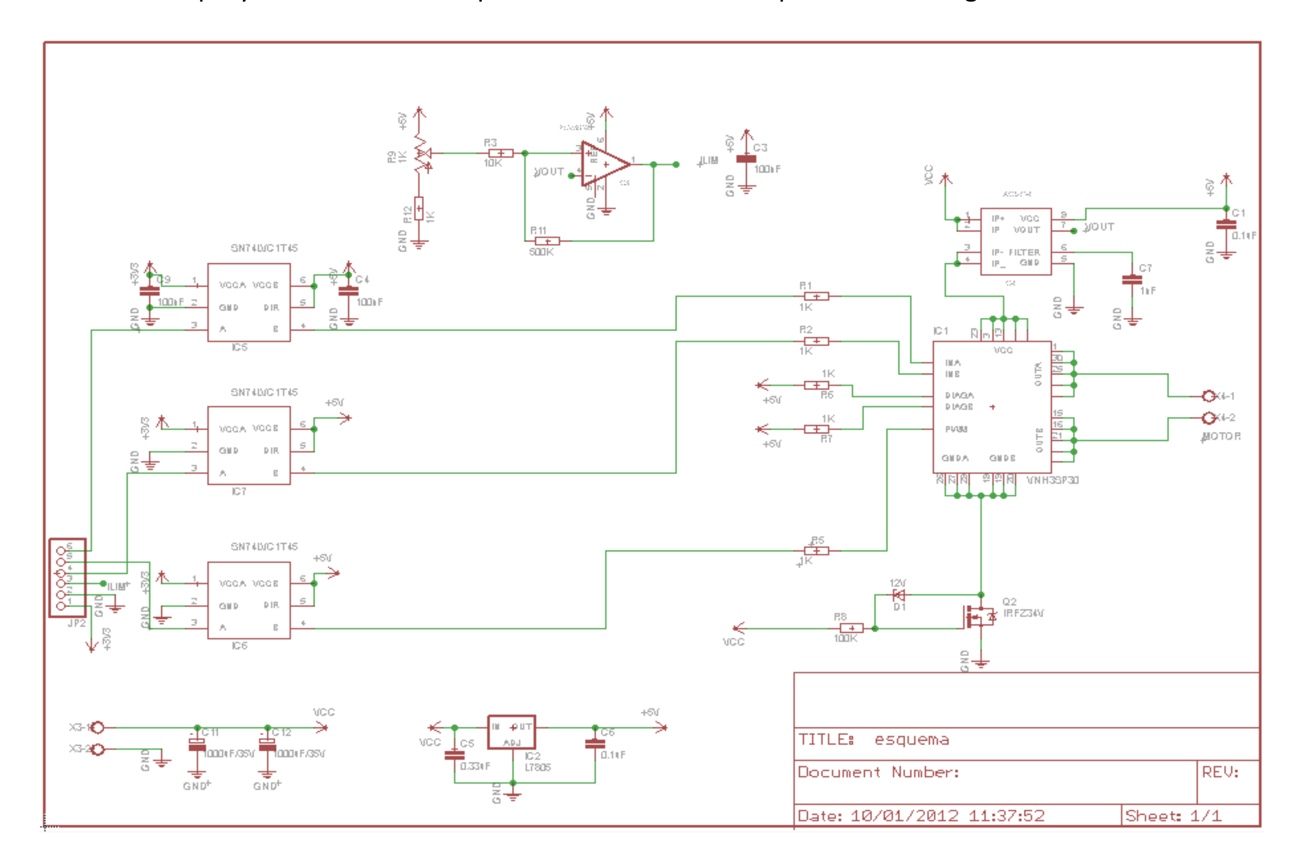


Los pines del header que se conecta a la FPGA se irán alternando y unas veces irán en la cara de arriba y otras en la de abajo, dependiendo de la disposición de la placa de potencia según la imagen anterior.

En el hueco que queda, delimitado por el conector de la FPGA entre placa de potencia y placa de potencia que están a la misma altura (1 y 3 en la figura), es donde se encajan todos los componentes THD que abultan, como son los condensadores, los MOSFET, etc. Por este motivo es por lo que queda tan alargada.

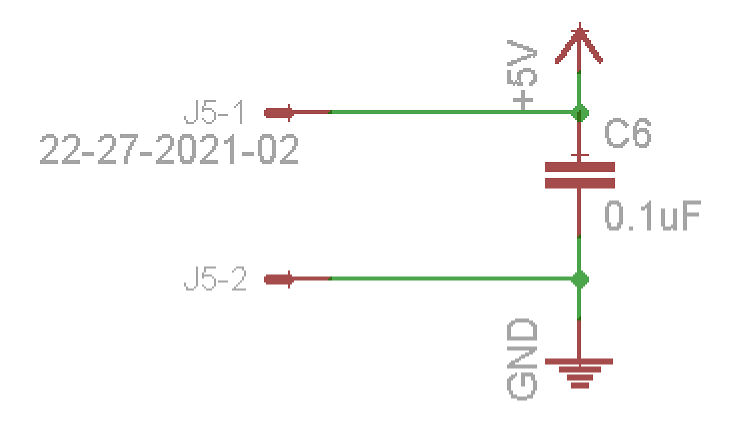
7.2 Mejoras

En el proyecto anterior el esquema del circuito había quedado de la siguiente manera.

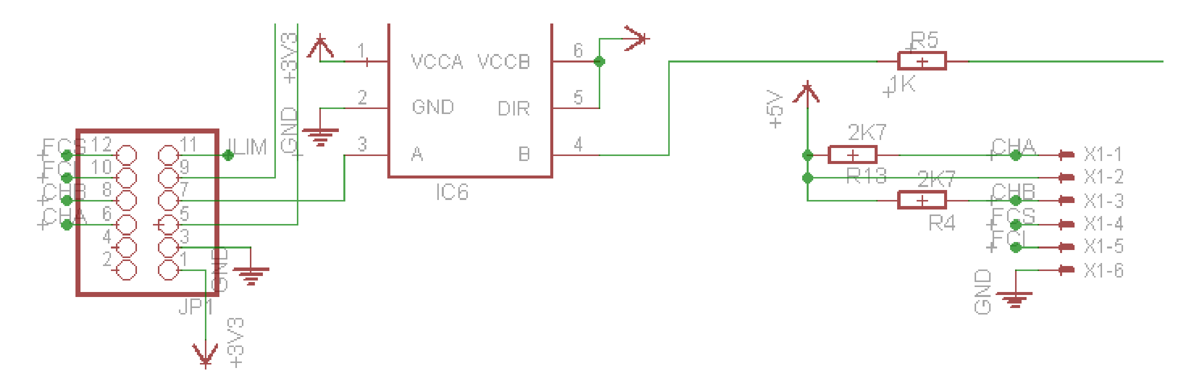


Había una errata en el circuito de protección contra inversión de polaridad, el transistor MOSFET tenía que estar al revés, es decir, Drenaje a masa y Source conectada a GND del VNH3SP 30-E, que ya está corregida.

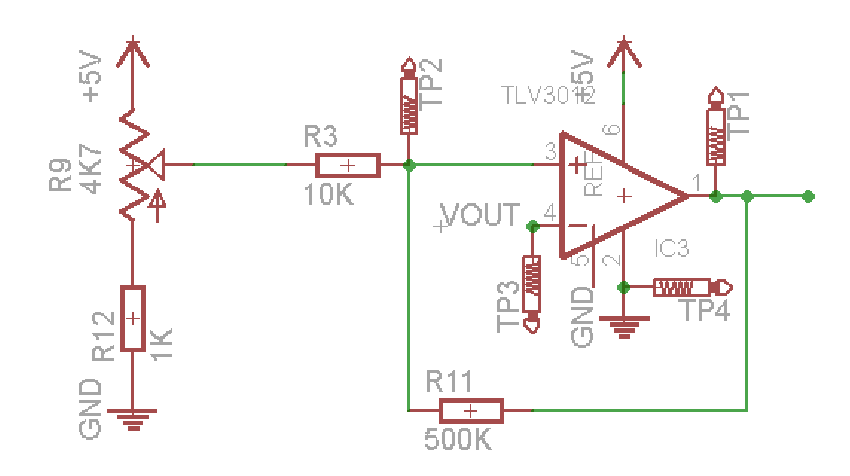
Para minimizar el tamaño la alimentación de 5 voltios se realiza con una fuente externa, en vez de con un regulador como venía implementado.



Para hacerla más compacta, los finales de carrera y el encoder se conectan a la FPGA a través de la placa de potencia. Y se añade una resistencia de pull-up de 2K7 ohmios.



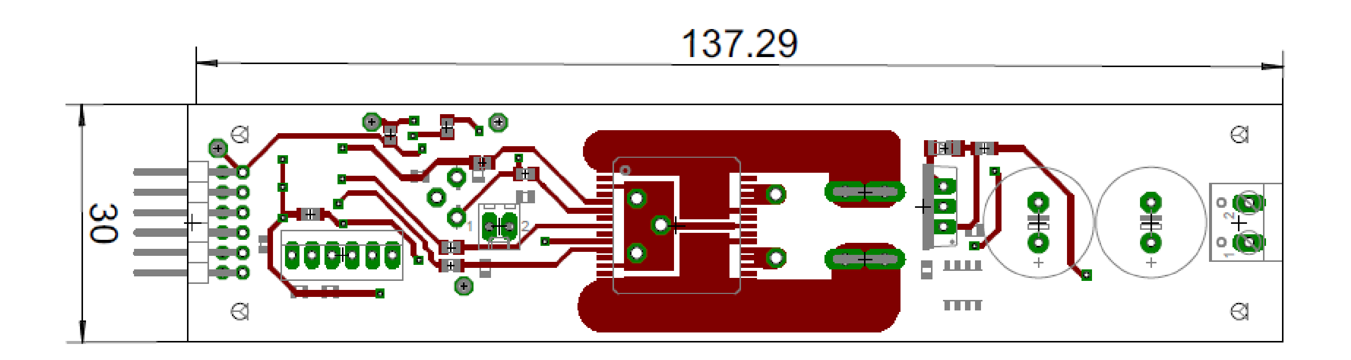
Y finalmente se introducen unos puntos de TEST para comprobar el funcionamiento del limitador de corriente, que, por desgracia, no funcionó con la primera placa del proyecto anterior.



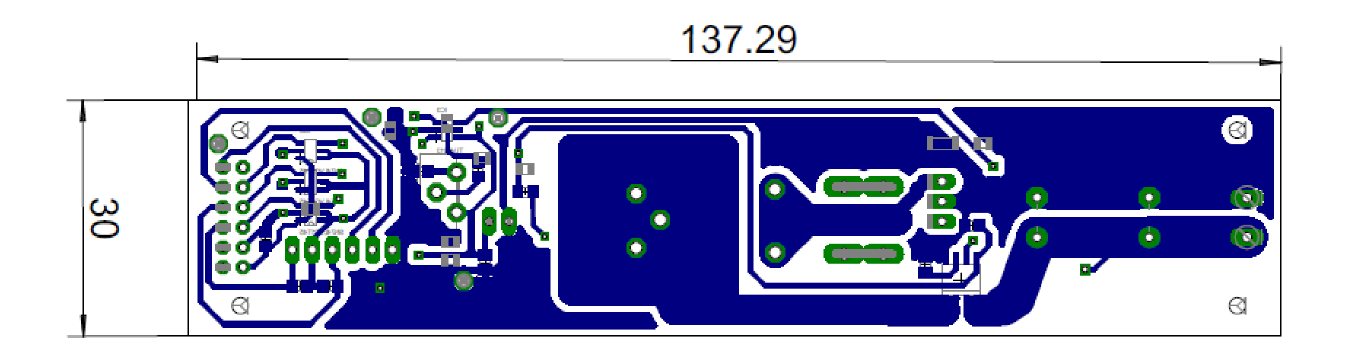
CAPÍTULO 8

8 Anexos

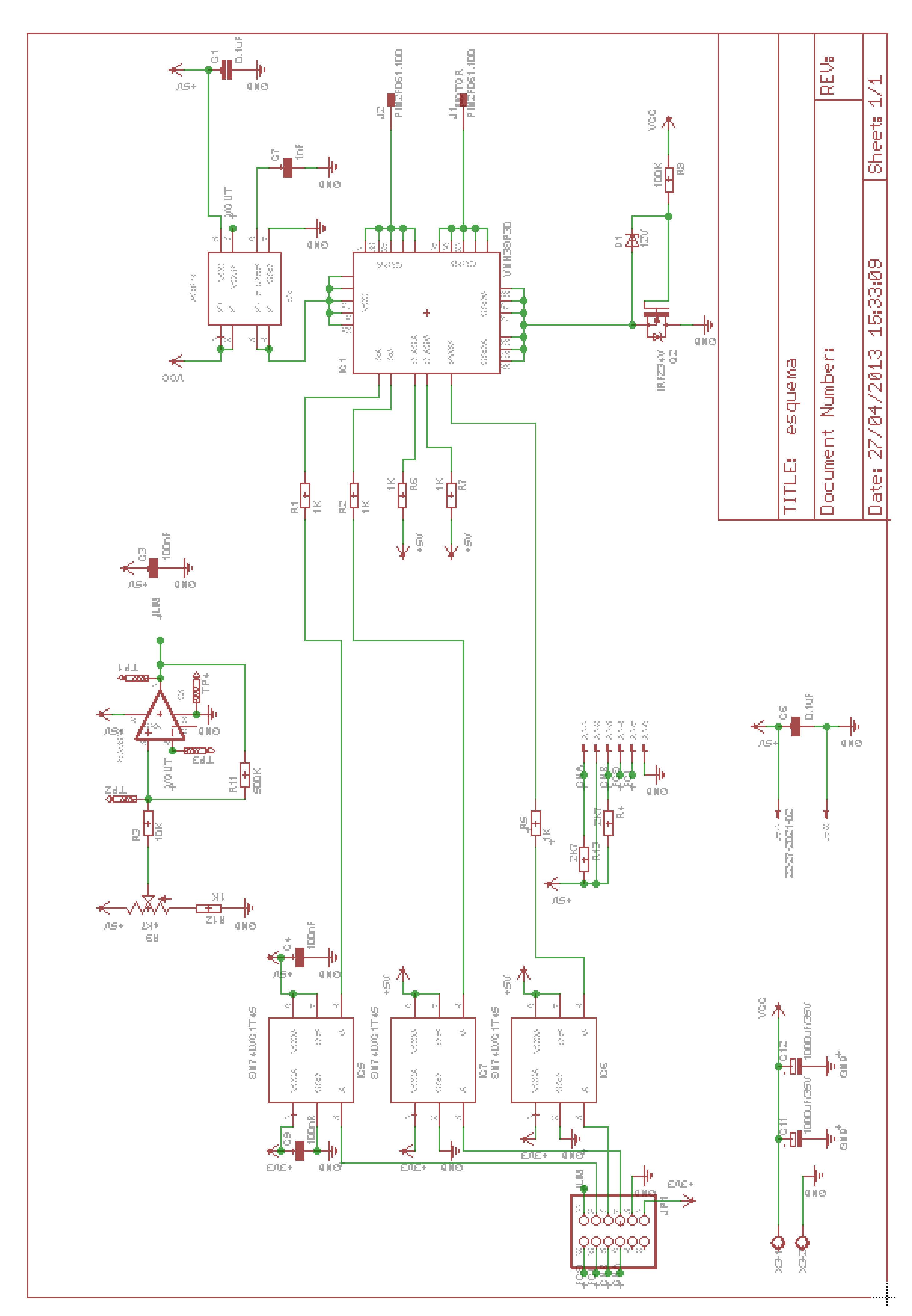
8.1 Planos



Cara top



Cara bottom



Esquema del circuito.

8.2 Manual de instrucciones



Switches y pulsadores de la FPGA.

Tal y como aparecen en la imagen, los switches están desactivados.

- SW7 o ILIM Disable, tiene que estar activado (en la imagen aparece desactivado), aún no se ha comprobado su funcionamiento.
- SW6 o FC Disable, desactivado, para que los finales de carrera sean reconocidos.
- SW5 no se usa.
- SW4 o BUCLE, si está tal como se indica en la imagen, el control del motor se realiza en bucle abierto. Si por el contrario está arriba, el control se realiza en bucle cerrado.

Para el control en **bucle abierto** se usan el resto de los switches y de los pulsadores. Mediante los pulsadores se define la velocidad a la que se moverá el motor. **Esta velocidad se refleja en el visualizador de 7 segmentos situado por encima de P1. Tiene 10 velocidades, desde la 0 en la que no se mueve, hasta la 9 que es la más rápida.**

- P3 o reset, inicializa todas las variables de la FPGA.
- P2 no se usa.
- P1 o DN, si es pulsado disminuye en 1 la velocidad del visualizador de 7 segmentos, si es 0 no disminuye más.
- P0 o UP, si es pulsado aumenta en 1 la velocidad del visualizador de 7 segmentos, si es 9 no aumenta más.

Los switches SW3, SW2, SW1 y SW0 se utilizan en bucle abierto, es decir con SW4 (BUCLE) abajo. Si SW4 está arriba es indiferente la posición de éstos.

- SW3 o CICLO, repite un ciclo consistente en subir y bajar 20 mm. continuamente. La velocidad de subida y bajada es la representada en el visualizador.

- SW2 o STEP, realiza una subida o bajada de 10 mm. cada vez que se pulsa UP (pulsador P0) o DN (pulsador P1).

- SW1 o BAJA hace bajar al actuador hasta que se desactiva o se alcanza FCI a la velocidad elegida con los pulsadores.

- SW0 o SUBE hace subir al actuador hasta que se desactiva o se alcanza FCS a la velocidad elegida con los pulsadores.

Para el control en **bucle cerrado** se usa el programa de MATLAB.

Se elige la trayectoria del motor, si es escalón o senoide. Se copia y pega el contenido en la interfaz de MATLAB.

Si es una senoide aparecerá el siguiente mensaje.

```
ingrese el valor de amplitud de la senoide =
ingrese el número de divisiones =
ingrese el punto de inicio (de 0 a 1000)=
```

Cuantas más divisiones tenga la senoide mejor se representará, pero más lento irá el motor. Cada división es un dato, y éstos son mandados como consigna al motor cada 10 ms. por lo que indirectamente se indica el periodo de la senoide.

El punto de inicio refleja la altura donde comienza el movimiento senoidal. 0 es en el final de carrera inferior y 1000, 10 cm. por encima, es decir, en el final de carrera superior.

Si es un escalón.

```
ingrese el valor de la altura del escalón =
ingrese el número de divisiones =
ingrese el punto de inicio =
```

Cuyo tiempo arriba y tiempo abajo es la mitad del periodo, y el periodo es el número de divisiones multiplicado por 10 milisegundos.

Tras introducir los datos se crea un archivo, llamado archivoconsigna.txt con las posiciones generadas que se mandan al motor.

Después de ser enviados todos los datos al motor se genera el siguiente mensaje.

```
INTRODUZCA EL NÚMERO.- 1 stop,2 start,3 mandar_posicion_on,4 mandar_posicion_off,5 volver_a_mandar,6 salir,7 ver posición =
```

Según el número introducido, se ejecutará una orden u otra. Es recomendable esperar a que el motor se encuentre en el estado de CONTROL, tras haberse ejecutado los anteriores estados, para introducir el número.

- 1 STOP, para el motor en la posición en la que se encuentre. La parada se realiza en bucle cerrado, por lo que el motor no puede moverse debido al peso o por cualquier otra circunstancia.
- 2 START, tras el estado STOP pone en marcha al motor, siguiendo la misma trayectoria que tenía antes de parar.
- 3 MANDAR_POS_ON no se usa.
- 4 MANDAR_POS_OFF no se usa.

  El punto 3 y 4 están implementados, por lo tanto funciona, pero no se recomienda su utilización.

- 5 VOLVER_A_ MANDAR, para el motor, y espera a que los nuevos datos sean introducidos. Se repite el programa y se muestra otra vez el mensaje de introducir datos para la senoide o para el escalón.
- 6 SALIR, se termina el programa.
- 7 ver posición, aparece el siguiente mensaje.

  ```
  ingrese el número de datos que quiere leer =
  ```

  Si se quiere leer sólo un periodo de la senoide o del escalón, se introduce el mismo número que se puso en el número de divisiones. Si no, se pueden leer tantos datos como se deseen. Al final se genera un archivo donde se guardan llamado archivoencoder.txt.

8.3 Código VHDL

8.3.1 TOP

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   17:56:21 12/15/2012
-- Design Name:
-- Module Name:   C:/Users/J/Desktop/vhdl/motordcproyect/TOPmotor_DC.vhd
-- Project Name:  motordcproyect
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: motor_DC
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY TOPmotor_DC IS
END TOPmotor_DC;

ARCHITECTURE behavior OF TOPmotor_DC IS

   -- Component Declaration for the Unit Under Test (UUT)
```

```
 COMPONENT motor_DC
 PORT(
     CLK : IN  std_logic;
     RST : IN  std_logic;
     UP : IN  std_logic;
     DN : IN  std_logic;
     ILIM : IN  std_logic;
     FCI : IN  std_logic;
     FCS : IN  std_logic;
     CHA : IN  std_logic;
     CHB : IN  std_logic;
     SW_SUBE : IN  std_logic;
     SW_BAJA : IN  std_logic;
     SW_STEP : IN  std_logic;
     SW_CICLO : IN  std_logic;
     SW_FC_DISABLE : IN  std_logic;
     SW_ILIM_DISABLE : IN  std_logic;
     BUCLE : IN  std_logic;
     RX : IN  std_logic;
     PWM : OUT  std_logic;
     INA : OUT  std_logic;
     INB : OUT  std_logic;
     LED_FCI : OUT  std_logic;
     LED_FCS : OUT  std_logic;
     SSEG : OUT  std_logic_vector(6 downto 0);
     AN : OUT  std_logic_vector(3 downto 0);
     DP : OUT  std_logic;
     TX : OUT  std_logic
    );
 END COMPONENT;


--Inputs
signal CLK : std_logic := '0';
signal RST : std_logic := '0';
signal UP : std_logic := '0';
signal DN : std_logic := '0';
signal ILIM : std_logic := '0';
signal FCI : std_logic := '0';
signal FCS : std_logic := '0';
signal CHA : std_logic := '0';
signal CHB : std_logic := '0';
signal SW_SUBE : std_logic := '0';
signal SW_BAJA : std_logic := '0';
```

```
   signal SW_STEP : std_logic := '0';
   signal SW_CICLO : std_logic := '0';
   signal SW_FC_DISABLE : std_logic := '0';
   signal SW_ILIM_DISABLE : std_logic := '0';
   signal BUCLE : std_logic := '1';
   signal RX : std_logic := '0';

 	--Outputs
   signal PWM : std_logic;
   signal INA : std_logic;
   signal INB : std_logic;
   signal LED_FCI : std_logic;
   signal LED_FCS : std_logic;
   signal SSEG : std_logic_vector(6 downto 0);
   signal AN : std_logic_vector(3 downto 0);
   signal DP : std_logic;
   signal TX : std_logic;

   -- Clock period definitions
   constant CLK_period : time := 20 ns;

BEGIN

 	-- Instantiate the Unit Under Test (UUT)
   uut: motor_DC PORT MAP (
          CLK => CLK,
          RST => RST,
          UP => UP,
          DN => DN,
          ILIM => ILIM,
          FCI => FCI,
          FCS => FCS,
          CHA => CHA,
          CHB => CHB,
          SW_SUBE => SW_SUBE,
          SW_BAJA => SW_BAJA,
          SW_STEP => SW_STEP,
          SW_CICLO => SW_CICLO,
          SW_FC_DISABLE => SW_FC_DISABLE,
          SW_ILIM_DISABLE => SW_ILIM_DISABLE,
          BUCLE => BUCLE,
          RX => RX,
          PWM => PWM,
          INA => INA,
```

```
        INB => INB,
        LED_FCI => LED_FCI,
        LED_FCS => LED_FCS,
        SSEG => SSEG,
        AN => AN,
        DP => DP,
        TX => TX
       );

   -- Clock process definitions
   CLK_process :process
   begin
                CLK <= '0';
                wait for CLK_period/2;
                CLK <= '1';
                wait for CLK_period/2;
   end process;


          RST<='1','0' after 20 ns;
   -- Stimulus process
   stim_proc: process
   begin
   UP <='0';
   DN <='0';
   ILIM <='1';
   FCI <='1';
   FCS <='1';
   CHA <= '0';
   CHB <= '0';
   SW_SUBE <= '0';
   SW_BAJA <= '0';
   SW_STEP <= '0';
   SW_CICLO <= '0';
   SW_FC_DISABLE <= '0';
   SW_ILIM_DISABLE <= '1';
   BUCLE <= '1';
   RX <= '1';
                --------------------------------PRUEBA DE RECEPCION----------------------------------------
----------


                --BUCLE<='1';
                wait for 1000 us;
```

```
RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00101111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00010101
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
```

```
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00001111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00011010
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
```

```
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00001111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00011111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
```

```
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00001111--MAS SIGNIFICATIVA DE 1001
mm--EL PRIMER BIT SE DESPRECIA
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI-------------------------ESTE ES EL ULTIMO QUE LEE(LOS DOS
ÚLTIMOS NO LOS LEE)
wait for 17600 ns;--1100*16--DATO 00011010-MENOS SIGNIFICATIVA DE 1001
mm-EL PRIMER BIT SE DESPRECIA
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
```

```
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 01000000-MENOS SIGNIFICATIVA DE 1001
mm-EL PRIMER BIT SE DESPRECIA
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 01000000--ENVIAR POS ENCODER
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
```

```
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

wait for 156000 us;

CHB<='1';
wait for 50 ns;
CHA<='1';
wait for 50 ns;
CHB<='0';
wait for 50 ns;
CHA<='0';
wait for 50 ns;
CHB<='1';
wait for 50 ns;
CHA<='1';
wait for 50 ns;
CHB<='0';
wait for 50 ns;
CHA<='0';
wait for 50 ns;
CHB<='1';
wait for 50 ns;
CHA<='1';
wait for 50 ns;
CHB<='0';
wait for 50 ns;
CHA<='0';
wait for 50 ns;
--FCI<='0';
RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 10000000
```

```
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

wait for 46000 us;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 01100000
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
```

```
wait for 17600 ns;

wait for 46000 us;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 11010110--NO ENVIAR POS ENCODER
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00111111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
```

```
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00011010
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00011111
RX <= '1';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
```

```
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00010010
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00010110
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
```

```
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00010010
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 00000000--MAS SIGNIFICATIVA DE 1001
mm--EL PRIMER BIT SE DESPRECIA
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
```

```
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';--MAS SIGNIFICATIVO
		wait for 17600 ns;
		RX <= '0';--PARIDAD
		wait for 17600 ns;
		RX <= '1';--ESTADO REPOSO
		wait for 17600 ns;

		RX <= '0';--BIT INI------------------------ESTE ES EL ULTIMO QUE LEE (LOS DOS
ÚLTIMOS NO LOS LEE)
		wait for 17600 ns;--1100*16--DATO 00001001-MENOS SIGNIFICATIVA DE 1001
mm-EL PRIMER BIT SE DESPRECIA
		RX <= '1';--MENOS SIGNIFICATIVO
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '1';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';
		wait for 17600 ns;
		RX <= '0';--MAS SIGNIFICATIVO
		wait for 17600 ns;
		RX <= '0';--PARIDAD
		wait for 17600 ns;
		RX <= '1';--ESTADO REPOSO
		wait for 17600 ns;

		RX <= '0';--BIT INI
		wait for 17600 ns;--1100*16--DATO 01000000-MENOS SIGNIFICATIVA DE 1001
mm-EL PRIMER BIT SE DESPRECIA
		RX <= '0';--MENOS SIGNIFICATIVO
		wait for 17600 ns;
		RX <= '0';
```

```
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 01000000-
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

FCI<='0';
wait for 26000 us;
```

```
RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 10000000-STOP
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;

wait for 26000 us;

RX <= '0';--BIT INI
wait for 17600 ns;--1100*16--DATO 01000000-START
RX <= '0';--MENOS SIGNIFICATIVO
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '0';
wait for 17600 ns;
RX <= '1';
wait for 17600 ns;
RX <= '0';--MAS SIGNIFICATIVO
wait for 17600 ns;
RX <= '1';--PARIDAD
```

```
wait for 17600 ns;
RX <= '1';--ESTADO REPOSO
wait for 17600 ns;




--motor descendiendo
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;
--CHB<='1';
--wait for 50 ns;
--CHA<='1';
--wait for 50 ns;

wait for 10 us;
FCI<='0';
wait for 100000 us;
```

```
                BUCLE<='0';

        --------------------------------PRUEBA DE ESTADOS--------------------------------

        --BUCLE2='0'=> ESTADO RESET
        --BUCLE2='1'=> ESTADO RESETBC
        BUCLE<='1';
        wait for 0.1 ms;
        --volvemos a RESET
        BUCLE<='0';
        FCI<='1';
        wait for 0.1 ms;
        --volvemos a RESETBC con el final de carrera inferior sin activar, por lo que iremos a
BAJARTOPE
        BUCLE<='1';
        wait for 0.1 ms;
        --Llegamos abajo por lo que se activa el final de carrera y vamos a AJUSTAR
        FCI<='0';
        wait for 0.1 ms;
        --Después del número de divisiones para ajustar vamos a CONTROL
        wait for 0.1 ms;
        --BUCLE='0'=> siguiente estado RESET
        BUCLE<='0';
        FCI<='0';
        wait for 0.1 ms;
        --volvemos a RESETBC con el fin de carrera activado
        BUCLE<='1';
        wait for 0.1 ms;
        --volvemos a CONTROL pasando por ajuste
        wait for 0.1 ms;

         -- wait for 100 ns;

    wait for CLK_period*10;

    wait;
  end process;

END;
```

8.3.2 MOTOR_DC

```
-- Control en bucle abierto
-- UART de 8 bits datos, 57600 baudios
-- A 115200 scilab no llega

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.STD_LOGIC_SIGNED.ALL;

entity motor_DC is
Port ( CLK : in STD_LOGIC;          --50MHz
  RST : in STD_LOGIC;
  UP : in STD_LOGIC;                --*SUBBLOQUE DETECTOR DE FLANCOS
  DN : in STD_LOGIC;                --*SUBBLOQUE DETERTOR DE FLANCOS
  ILIM :in STD_LOGIC;               --CORRIENTE EXCESIVA, activo L        *MAQUINA DE ESTADOS
  FCI:in STD_LOGIC;                 --FINAL DE CARRERA INFERIOR, activo L    *MAQUINA DE
ESTADOS
  FCS:in STD_LOGIC;                 --FINAL DE CARRERA SUPERIOR, activo L              *MAQUINA
DE ESTADOS
  CHA: in STD_LOGIC;                --CANAL A ENCODER                              *SUBBLOQUE
ENCODER
  CHB : in STD_LOGIC;               --CANAL B ENCODER                              *SUBBLOQUE
ENCODER
  SW_SUBE:in STD_LOGIC;             --Sube hasta FCS
  SW_BAJA: in STD_LOGIC;            --Baja hasta FCI e inicializa CNT_ENC al llegar
  SW_STEP: in STD_LOGIC;            --Sube o baja una distancia fija al pulsar UP o DN (para test)
  SW_CICLO: in STD_LOGIC;           --Ciclo de subidas y bajadas sin fin
  SW_FC_DISABLE: in STD_LOGIC; --Deshabilita los finales de carrera Â¡MUY PELIGROSO!
  SW_ILIM_DISABLE: in STD_LOGIC; --Deshabilita limitacion de intensidad
  BUCLE: in STD_LOGIC; --Elegir bucle abierto '0' o cerrado '1'---------------------------------------------
---
  RX :in STD_LOGIC;--Recepción serie---------------------------------------------------------------------------
  PWM: out STD_LOGIC;               --*SALIDA PWM
  INA: out STD_LOGIC;               --*SENTIDO DE GIRO DEL MOTOR
  INB: out STD_LOGIC;               --*SENTIDO DE GIRO DEL MOTOR
  LED_FCI: out STD_LOGIC;
  LED_FCS: out STD_LOGIC;
  --LED0: out STD_LOGIC;
  --LED1: out STD_LOGIC;
  --LED2: out STD_LOGIC;
  LED3: out STD_LOGIC;
  LED4: out STD_LOGIC;
```

```
  LED5: out STD_LOGIC;
  LED6: out STD_LOGIC;
  LED7: out STD_LOGIC;
  SSEG : out STD_LOGIC_VECTOR (6 downto 0);
  AN : out STD_LOGIC_VECTOR (3 downto 0);
  DP : out STD_LOGIC;
  TX : out STD_LOGIC);
end motor_DC;

architecture Behavioral of motor_DC is
  signal UP1, UP2, UP3, DN1, DN2, DN3: std_logic;
  signal BUCLE1,BUCLE2: std_logic;-------------------------------------------------------------------------------
-------------
  ---------------------------------------------RAM-------------------------------------------------------------------------
----
  CONSTANT ADDRESS_WIDTH : integer :=12; --14;--1minuto=>60segundos=>(a
10ms)=>6000posiciones=>(1 posicion con dos vectores de 5bit)=>12000=>2^14
  CONSTANT      DATA_WIDTH    : integer := 5;
  signal data:std_logic_vector(DATA_WIDTH - 1 DOWNTO 0);
  signal we         :std_logic;
  TYPE MRAM IS ARRAY(0 TO 2 ** ADDRESS_WIDTH - 1) OF std_logic_vector(DATA_WIDTH - 1
DOWNTO 0);
  SIGNAL ram_block : MRAM;
  -----------------------------------------------------------------------------------------------------------------------------
-
  signal nSALIDA_RAM,SALIDA_RAM : signed(13 DOWNTO 0);---------------------------------------------
---------------------------------
  signal nSINCRONIZADOR_RAM,SINCRONIZADOR_RAM : signed(13 DOWNTO 0);-------------------
-----------------------------------------------------------
  signal CONSIGNA : signed(13 DOWNTO 0);-----------------------------------------------------------------------
-------
  signal nCONSIGNAx256,CONSIGNAx256 : signed(21 DOWNTO 0);---------------------------------------
------------------------------------
  signal nCONSIGNABAJARx256,CONSIGNABAJARx256 : signed (21 DOWNTO 0);---------------------
------------------------------------
  signal nCNT13,CNT13: std_logic_vector(10 DOWNTO 0);-------------------13 segundos
  signal CNT_WRITE_RAM, nCNT_WRITE_RAM,CNT_READ_RAM,
nCNT_READ_RAM,CNT_ENVIAR,nCNT_ENVIAR,ULTIMA_POS: std_logic_vector
(ADDRESS_WIDTH - 1 DOWNTO 0);-------------
  signal E_PARIDAD ,E_FORMATO ,E_SOBRESCRITURA :std_logic;--No se usan------------------------
---------------------------------
  signal FCI1, FCI2, FCS1, FCS2, ILIM1, ILIM2: std_logic;
  signal CHA1, CHA2, CHA3, CHB1, CHB2, CHB3: std_logic;
  signal SW_SUBE2, SW_SUBE1, SW_BAJA2, SW_BAJA1: std_logic;
```

```
signal SW_STEP2, SW_STEP1, SW_CICLO2, SW_CICLO1: std_logic;
signal DIV_10K, nDIV_10K: std_logic_vector (12 downto 0);     --Genera CE_10K de 10 kHz
signal DIV_100K,nDIV_100K: std_logic_vector (8 downto 0);     --Genera CE_100K de 100 kHz
signal DIV_100, nDIV_100: std_logic_vector (6 downto 0);      --Genera CE_100 de 100 Hz a
partir de CE_10K
signal DIV_200, nDIV_200: std_logic_vector (5 downto 0);      --Genera CE_200 de 200 Hz a
partir de CE_10K------------------
signal DIV_TX, nDIV_TX: std_logic_vector (9 downto 0);        --Genera EN_CLKTx de 115200
Hz / 57600 Hz
signal DIV_RX, nDIV_RX: std_logic_vector (5 downto 0);        --Genera EN_CLKRx 16 veces
más rápido que 115200 Hz / 57600 Hz
signal CNT_TIME, nCNT_TIME: std_logic_vector (14 downto 0);          --Contador de periodos
de 100 Hz desde el reset (5 min)
signal CNT_REF, nCNT_REF: std_logic_vector (6 downto 0);-------------------------
MODIFICADO(MAS GRANDE)----------------------
signal CNT_PWM, nCNT_PWM: std_logic_vector (6 downto 0);-------------------------ESTE
TAMBIEN--------------------------------
signal CNT_ENC, nCNT_ENC: std_logic_vector(15 downto 0);--AÑADO 1 MAS
signal OPERACION1,OPERACION2: signed(13 downto 0);--------------------------------------------------
----
signal POS_DESEADA,OPERACION3,OPERACION4: signed(14 downto 0);--+1
signal DIFERENCIA,ERROR,ERROR_ACOTADO: signed(15 downto 0);---+1
signal FINS_CNT_ENC, nFINS_CNT_ENC: std_logic_vector(15 downto 0);--+1
signal FINB_CNT_ENC, nFINB_CNT_ENC: std_logic_vector(15 downto 0);--+1
type TIPO_IU_ESTADO is (ESPERAR,RESETBC,BAJAR_TOPE,AJUSTAR,CONTROL,RESET,
INI_CICLO, PARO_I, SUBIDA, PARO_S, BAJADA);--Añadidos------
signal IU_ESTADO, nIU_ESTADO: TIPO_IU_ESTADO;
type TIPO_UART_ESTADO is
(LEERUNAVEZ,NADA,LEERDATOS,ACABANDATOSRX,STOP,MANDARPOSENC_ON,MANDARPOSE
NC_OFF,PARAR);
signal UART_ESTADO, nUART_ESTADO: TIPO_UART_ESTADO;
signal BCD: std_logic_vector (6 downto 0);-------------------------------MODIFICADO-------------------
-----------------------
signal CE_10K,CE_100K, CE_100,CE_200, EN_CLKTx,EN_CLKRx,RD: std_logic;
signal FUP, FDN, PULSO_CHA: std_logic;
signal SEPUEDEENVIAR,nSEPUEDEENVIAR:std_logic;
-- Para el envio serie de datos por la UART
signal WR_TX, TX_RDY,RX_RDY,RX_RDY1,RX_RDY2,RX_RDY3: std_logic;------------INTRODUZCO
RX_RDY
signal DTX,DRX: std_logic_vector (7 downto 0);-----------------------------------INTRODUZCO DRX
type TIPO_TX_ESTADO is (RESET, ESPERA, ENVIA, INCREMENTA);
signal TX_ESTADO, nTX_ESTADO: TIPO_TX_ESTADO;
type TIPO_REG_DTX is array (natural range <>) of std_logic_vector(7 downto 0);
signal REG_DTX, nREG_DTX: TIPO_REG_DTX(0 to 5);
```

```
 signal CNT_REG_DTX, nCNT_REG_DTX: integer range 0 to 5;
 signal LD3,LD4,LD5,LD6,LD7,NLD3,NLD4,NLD5,NLD6,NLD7: std_logic;

	COMPONENT UART_TX
	generic (DWL1: integer range 0 to 7 := 7); -- UART con 8 bits de datos
	PORT(
		CLK : IN std_logic;
		EN : IN std_logic;
		RST : IN std_logic;
		WR : IN std_logic;
		DIN : IN std_logic_vector(DWL1 downto 0);
		Tx : OUT std_logic;
		TxRDY : OUT std_logic
		);
	END COMPONENT;

	COMPONENT UART_RX
	PORT (
		CLK  : in std_logic;
		EN   : in std_logic;
		RST  : in std_logic;
		Rx   : in std_logic;
		RD   : in std_logic;
		DOUT : out std_logic_vector(7 downto 0);
		ERROR_PARIDAD  : out std_logic;
		ERROR_FORMATO  : out std_logic;
		SOBRESCRITURA  : out std_logic;
		RxRDY: out std_logic
		);
	END COMPONENT;

begin

Inst_UART_Tx: UART_TX PORT MAP(
	CLK => CLK,
	EN => EN_CLKTx,
	RST => RST,
	WR => WR_TX,
	DIN => DTX,
	Tx => TX,
	TxRDY => TX_RDY
);

Inst_UART_Rx: UART_RX PORT MAP(
```

```
            CLK  => CLK,
            EN   => EN_CLKRx ,
            RST  => RST ,
            Rx   => RX ,
            RD   => RD ,
            DOUT => DRX ,
            ERROR_PARIDAD  => E_PARIDAD ,
            ERROR_FORMATO  => E_FORMATO ,
            SOBRESCRITURA  => E_SOBRESCRITURA ,
            RxRDY => RX_RDY
);

LED_FCI <= FCI2;
LED_FCS <= FCS2;
LED3<=LD3;
LED4<=LD4;
LED5<=LD5;
LED6<=LD6;
LED7<=LD7;

-- Proceso secuencial
process (CLK,RST)
begin
  if (RST='1') then
    UP1 <= '0'; UP2 <= '0'; UP3 <= '0';
    DN1 <= '0'; DN2 <= '0'; DN3 <= '0';
            BUCLE1<= '0'; BUCLE2<='0';--------------------------------------------------------------------------------
---------------
            CNT_WRITE_RAM<=(others=>'0');-----------------------------------------------------------------------
---------------------
            CNT_READ_RAM<=(others=>'0');-------------------------------------------------------------------------
--------------------
            CNT_ENVIAR<=(OTHERS=>'0');
            SALIDA_RAM<=(others=>'0');------------------------------------------------------------------------------
-----------------
            SINCRONIZADOR_RAM<=(others=>'0');-----------------------------------------------------------------
------------------------------
            CONSIGNAx256<=(others=>'0');--------------------------------------------------------------------------
---------------------
            CONSIGNABAJARx256<="0010011110001000000000";--2530+8ceros=>
"111111111111111"=32767/12=2730-200=2530=>Deshaciendo el bucle
            CNT13<=(others=>'0');--------------------------------------------------------------------------------------
-------------
    FCI1 <= '0'; FCI2 <= '0';
```

```
   FCS1 <= '0'; FCS2 <= '0';
   ILIM1 <= '0'; ILIM2 <= '0';
   CHA1<='0'; CHA2<='0'; CHA3<='0';
   CHB1<='0'; CHB2<='0'; CHB3<='0';
          RX_RDY1<='0';RX_RDY2<='0';RX_RDY3<='0';----------------------------------------------------------
------------------------
   SW_SUBE2<='0'; SW_SUBE1<='0';
   SW_BAJA2<='0'; SW_BAJA1<='0';
   SW_STEP2<='0'; SW_STEP1<='0';
   SW_CICLO2<='0'; SW_CICLO1<='0';
   DIV_10K <= (others=>'0');
          DIV_100K <= (others=>'0');---------------------------------------------------------------------------------
---------------
   DIV_100 <= (others=>'0');
          DIV_200 <= (others=>'0');----------------------------------------------------------------------------------
----------------
   DIV_TX <= (others=>'0');
          DIV_RX <= (others=>'0');
   CNT_TIME <= (others=>'0');
   CNT_REF <= (others=>'0');
   CNT_PWM <= (others=>'0');
   CNT_ENC <= "0111111111111111"; --DespuÃ©s del reset descender actuador hasta FCI,
empieza en 0 para que no se desborde por arriba
   FINS_CNT_ENC <= "0111111111111111";
   FINB_CNT_ENC <= (others=>'0');
   CNT_REG_DTX <= 0;
   TX_ESTADO<=RESET;
   IU_ESTADO<=RESET;------------------------------------------------------------------------------------------
          UART_ESTADO<=LEERUNAVEZ;
          SEPUEDEENVIAR<='1';
          LD3<='0';
          LD4<='0';
          LD5<='0';
          LD6<='0';
          LD7<='0';

  elsif (CLK'event and CLK='1') then --Registros

          CNT_WRITE_RAM<=nCNT_WRITE_RAM;---------------------------------------------------------------
----------------------------
          CNT_ENVIAR<=nCNT_ENVIAR;
   FCI2<=FCI1; FCI1<=FCI or SW_FC_DISABLE;
   FCS2<=FCS1; FCS1<=FCS or SW_FC_DISABLE;
   ILIM2<=ILIM1; ILIM1<=ILIM or SW_ILIM_DISABLE;
```

```
   CHA3<=CHA2; CHA2<=CHA1; CHA1<=CHA;
   CHB3<=CHB2; CHB2<=CHB1; CHB1<=CHB;
         RX_RDY3<=RX_RDY2;RX_RDY2<=RX_RDY1;RX_RDY1<=RX_RDY;--------------------------------
---------------------------------------
   DIV_10K <= nDIV_10K;
         DIV_100K <= nDIV_100K;
         DIV_TX <= nDIV_TX;
         DIV_RX <= nDIV_RX;
   CNT_REG_DTX <= nCNT_REG_DTX;
   TX_ESTADO <= nTX_ESTADO;
   IU_ESTADO <= nIU_ESTADO;
         UART_ESTADO <= nUART_ESTADO;
   CNT_ENC <= nCNT_ENC;
   FINS_CNT_ENC <= nFINS_CNT_ENC;   --Limite de la subida
   FINB_CNT_ENC <= nFINB_CNT_ENC;  --Limite de la bajada
         SINCRONIZADOR_RAM<=nSINCRONIZADOR_RAM;
         SEPUEDEENVIAR<=nSEPUEDEENVIAR;
         LD3<=NLD3;
         LD4<=NLD4;
         LD5<=NLD5;
         LD6<=NLD6;
         LD7<=NLD7;

         ----------------------------------------------------------
         if (CE_100K='1') then --Registros con habilitacion de 100 kHz--------Modificado
   CNT_PWM <= nCNT_PWM;
         end if;

         -----------------------------------------------------------

   if (CE_10K='1') then --Registros con habilitacion de 10 kHz
          CNT_REF <= nCNT_REF;
     UP3 <= UP2; UP2 <= UP1; UP1 <= UP;
          BUCLE2<=BUCLE1; BUCLE1<=BUCLE;----------------------------------------------------------------
-----------------------
     DN3 <= DN2; DN2 <= DN1; DN1 <= DN;
     SW_SUBE2 <= SW_SUBE1; SW_SUBE1 <= SW_SUBE;
     SW_BAJA2 <= SW_BAJA1; SW_BAJA1 <= SW_BAJA;
     SW_STEP2 <= SW_STEP1; SW_STEP1 <= SW_STEP;
     SW_CICLO2 <= SW_CICLO1; SW_CICLO1 <= SW_CICLO;
     DIV_100 <= nDIV_100;
          DIV_200 <= nDIV_200;
   end if;
   if (CE_100='1') then --Registros con habilitacion de 100 Hz
```

```
            SALIDA_RAM<=nSALIDA_RAM;---------------------------------------------------------------------
------------------------
            CONSIGNAx256<=nCONSIGNAx256;-----------------------------------------------------------------
----------------------------
            CONSIGNABAJARx256<=nCONSIGNABAJARx256;---------------------------------------------------
-----------------------------------------
     CNT13<=nCNT13;
            CNT_TIME <= nCNT_TIME;
     REG_DTX <= nREG_DTX;
   end if;
          if (CE_200='1') then --Registros con habilitacion de 200 Hz
            ----------------------------------------------------------------------------------
            CNT_READ_RAM<=nCNT_READ_RAM;---------------------------------------------------------------
---------------------------
          end if;

  end if;
end process;

-- Clock enable 10 kHz
process(DIV_10K)
begin
  if (DIV_10K = 4999) then
    nDIV_10K <= (others=>'0');
  else
    nDIV_10K <= DIV_10K + 1;
  end if;
end process;
CE_10K <= '1' when (DIV_10K=4999) else '0';

-- Clock enable 100 kHz
process(DIV_100K)
begin
  if (DIV_100K = 499) then
    nDIV_100K <= (others=>'0');
  else
    nDIV_100K <= DIV_100K + 1;
  end if;
end process;
CE_100K <= '1' when (DIV_100K=499) else '0';----------------------------Añadido para la frecuencia
de 100 KHz----------------

-- Clock enable 100 Hz (DIV_100 se habilita con CE_10K)
process(DIV_100)
```

```
begin
  if (DIV_100 = 99) then
    nDIV_100 <= (others=>'0');
  else
    nDIV_100 <= DIV_100 + 1;
  end if;
end process;
CE_100 <= '1' when (DIV_100=99) and (CE_10K='1') else '0';

-- Clock enable 200 Hz (DIV_200 se habilita con CE_10K)--5 ms
process(DIV_200)
begin
  if (DIV_200 = 49) then
    nDIV_200 <= (others=>'0');
  else
    nDIV_200 <= DIV_200 + 1;
  end if;
end process;
CE_200 <= '1' when (DIV_200=49) and (CE_10K='1') else '0';

-- Clock enable RX

process (DIV_RX)
begin
  if (DIV_RX = 53) then
    nDIV_RX <= (others=>'0');
  else
    nDIV_RX <= DIV_RX + 1;
  end if;
end process;

EN_CLKRx <= '1' when (DIV_RX=0) else '0';

-- Clock enable TX (16 veces más lento que el de RX)

process (DIV_TX,EN_CLKRx)
begin
  if (DIV_TX = 16) then
    nDIV_TX <= (others=>'0');
  elsif EN_CLKRx='1' then
    nDIV_TX <= DIV_TX + 1;
  else nDIV_TX<=DIV_TX;
  end if;
end process;
```

```
EN_CLKTx <= '1' when (DIV_TX=15) and (EN_CLKRx='1') else '0';

-- Contador de tiempo (ciclos de 100 Hz) modulo 30000 (5 min.) para envÃ-o serie del instante
actual
process(CNT_TIME)
begin
  if (CNT_TIME = 29999) then
    nCNT_TIME <= (others=>'0');
  else
    nCNT_TIME <= CNT_TIME + 1;
  end if;
end process;

-- Indice del registro REG_DTX a enviar por la UART
process(CNT_REG_DTX,TX_ESTADO,CNT_ENVIAR,ULTIMA_POS)
begin
  if (TX_ESTADO = RESET) then
    nCNT_REG_DTX <= 0;
  elsif (TX_ESTADO = INCREMENTA) then
    nCNT_REG_DTX <= CNT_REG_DTX + 1;
          if CNT_ENVIAR=ULTIMA_POS then
                  nCNT_ENVIAR<=(others=>'0');
          else
                  nCNT_ENVIAR<=CNT_ENVIAR+1;
          end if;
  else
    nCNT_REG_DTX <= CNT_REG_DTX;
          nCNT_ENVIAR<=CNT_ENVIAR;
  end if;
end process;

-- Registro de datos a enviar por la UART
nREG_DTX(0) <= "00"&nCNT_ENC(13 downto 8);
nREG_DTX(1) <= nCNT_ENC(7 downto 0);
nREG_DTX(2) <= "0000"&nCNT_READ_RAM(11 downto 8);
nREG_DTX(3) <= nCNT_READ_RAM(7 downto 0);
nREG_DTX(4) <= "0000000"&NLD7;
nREG_DTX(5) <= "00000000";

-- Multiplexor del dato a enviar por la UART
DTX <= REG_DTX(0) when (CNT_REG_DTX=0) else
      REG_DTX(1) when (CNT_REG_DTX=1) else
      REG_DTX(2) when (CNT_REG_DTX=2) else
```

```
      REG_DTX(3) when (CNT_REG_DTX=3) else
      REG_DTX(4) when (CNT_REG_DTX=4) else
      REG_DTX(5);

-- Máquina de estados para controlar UART_TX
process(TX_ESTADO,CE_100,EN_CLKTx,TX_RDY,CNT_REG_DTX,SEPUEDEENVIAR)-------------------
-----------------------------------------
begin
  nTX_ESTADO <= TX_ESTADO;
  WR_TX<='0';
  case TX_ESTADO is
    when RESET =>
      if (CE_100='1') and (SEPUEDEENVIAR='1') then nTX_ESTADO <= ESPERA;--------------------------
---------------------------------------
      end if;
    when ESPERA =>
      if (TX_RDY='1') then nTX_ESTADO <= ENVIA;
      end if;
    when ENVIA =>
      WR_TX<='1';
      if (EN_CLKTx='1') then nTX_ESTADO <= INCREMENTA;
      end if;
    when INCREMENTA =>
      if (CNT_REG_DTX=5) then nTX_ESTADO <= RESET;
      else nTX_ESTADO <= ESPERA;
      end if;
    when others =>
      nTX_ESTADO <= RESET;
  end case;
end process;

-- Detector flancos de subida de los pulsadores
FUP <= UP2 and not UP3;
FDN <= DN2 and not DN3;
PULSO_CHA <= CHA2 and not CHA3;

-- Contador del periodo del pwm (periodo 0.1 ms)---------------------------------------MODIFICADO
process(CNT_PWM)
begin
  if (CNT_PWM=90) then
    nCNT_PWM <= (others=>'0');
  else
    nCNT_PWM <= CNT_PWM + 1;
  end if;
```

```
end process;

-- Contador de referencia del pwm (0-90)-----------------------Aumentamos de 10 en 10 en vez de
en 1 en 1
process(FUP, FDN, CNT_REF, SW_STEP2)
begin
  if (FUP='1' and FDN='0' and CNT_REF<100) then -- UP
    nCNT_REF <= CNT_REF + 10;
  elsif (FUP='0' and FDN='1' and CNT_REF>0) then -- DOWN
    nCNT_REF <= CNT_REF - 10;
  else
    nCNT_REF <= CNT_REF;--Asignacion por defecto
  end if;
  if (SW_STEP2='1') then --Deshabilita FUP y FDN para la referencia
    nCNT_REF <= CNT_REF;
  end if;
end process;

-- Comparador de PWM (en el bucle cerrado)

BCD <= CNT_REF;

-- Decodificador de 7 segmentos-------------------------------Modificado al haber aumentado de 10
en 10
SSEG <= "0000001" when (BCD = 0) else
        "1001111" when (BCD = 10) else
        "0010010" when (BCD = 20) else
        "0000110" when (BCD = 30) else
        "1001100" when (BCD = 40) else
        "0100100" when (BCD = 50) else
        "0100000" when (BCD = 60) else
        "0001111" when (BCD = 70) else
        "0000000" when (BCD = 80) else
        "0000100";
AN <= "1110";
DP <= '1';

------------------------ ENCODER ----------------------------------------------
-- Encoder 360 pulsos/revoluciÃ³n y actuador 3 mm de paso => 120 pulsos/mm => 12
pulsos/décima de mm.

process (PULSO_CHA, CNT_ENC, CHB3,IU_ESTADO,FCI2)
begin
  nCNT_ENC <= CNT_ENC;
```

```
  if (IU_ESTADO=RESET) or (IU_ESTADO=RESETBC) then--tambien en RESETBC por posibles
desbordamiento debido a la inercia del motor ya que por RESET pasa en un momento
  nCNT_ENC<="0111111111111111";
  elsif IU_ESTADO=ESPERAR and (FCI2='0') then
  nCNT_ENC<= "0000100101100000";-------posicion (0+200)* 12----(20 mm desde pos abajo
para que no desborde)
  elsif (PULSO_CHA='1') then
    if (CHB3='0') then
      nCNT_ENC <= CNT_ENC + 1; --sube
    else
      nCNT_ENC <= CNT_ENC - 1; --baja
    end if;
  end if;
end process;

------------------------------------------------------------PWM BUCLE CERRADO

-- Máquina de estados para el control del motor según las instrucciones recibidas
process
(DRX,UART_ESTADO,SINCRONIZADOR_RAM,SEPUEDEENVIAR,SALIDA_RAM,CNT_WRITE_RAM,
FCS2,FCI2,IU_ESTADO)
begin
nSALIDA_RAM<=SINCRONIZADOR_RAM;
nUART_ESTADO <= UART_ESTADO;
nSEPUEDEENVIAR<=SEPUEDEENVIAR;

  case UART_ESTADO is
    when LEERUNAVEZ=>
			if DRX (7 downto 5)="001" then
				nUART_ESTADO<=LEERDATOS;
			end if;
    when NADA =>
			case DRX (7 downto 5) is
				when "001" =>
					nUART_ESTADO<=LEERDATOS;
				when "100" =>
					nUART_ESTADO<=STOP;
				when "101" =>
					nUART_ESTADO<=MANDARPOSENC_ON;
				when "110" =>
					nUART_ESTADO<=MANDARPOSENC_OFF;
				when "111" =>
					nUART_ESTADO<=PARAR;
				when others =>
```

```
                                if (FCI2='0' or FCS2='0') and (IU_ESTADO=CONTROL) then
        nUART_ESTADO <= STOP;
                                elsif (FCS2='0') and (IU_ESTADO=AJUSTAR) then
nUART_ESTADO <= STOP;
                                else nUART_ESTADO <= UART_ESTADO;
                                end if;
                end case;
        when PARAR =>
                nUART_ESTADO<=NADA;
        when LEERDATOS =>
                nSALIDA_RAM<=SALIDA_RAM;
                if DRX (7 downto 5)="010" then
                        nUART_ESTADO<=ACABANDATOSRX;
                end if;
        when ACABANDATOSRX =>
                ULTIMA_POS<=CNT_WRITE_RAM;
                nUART_ESTADO<=NADA;
        when STOP =>
                nSALIDA_RAM<=SALIDA_RAM;
                if (DRX (7 downto 5)="011") or (IU_ESTADO=RESET) then
                        nUART_ESTADO<=NADA;
                elsif (DRX (7 downto 5)="111") then
                        nUART_ESTADO<=PARAR;
                end if;
        when MANDARPOSENC_ON =>
                nSEPUEDEENVIAR<='1';
                nUART_ESTADO<=NADA;
        when MANDARPOSENC_OFF =>
                nSEPUEDEENVIAR<='0';
                nUART_ESTADO<=NADA;
        when others =>
                nUART_ESTADO<=UART_ESTADO;
 end case;

end process;

-----------------------------------------------------------------Control de ram
we <= RX_RDY2 and not RX_RDY3;

--contadores
process (CNT_WRITE_RAM,CNT_READ_RAM,we,IU_ESTADO,CE_100,UART_ESTADO)
begin
nCNT_WRITE_RAM<=CNT_WRITE_RAM;
nCNT_READ_RAM<=CNT_READ_RAM;
```

```
--CONTADOR ESCRITURA EN LA RAM

	if  IU_ESTADO=RESET then
		nCNT_WRITE_RAM<=(others=>'0');
		nCNT_READ_RAM<=(others=>'0');
	elsif (we='1') and UART_ESTADO=LEERDATOS then
		if CNT_WRITE_RAM<(2 ** ADDRESS_WIDTH - 1) then----------------evitamos
desbordamiento
			nCNT_WRITE_RAM<=CNT_WRITE_RAM+1;
		else nCNT_WRITE_RAM<=CNT_WRITE_RAM;
		end if;
	end if;

--CONTADOR LECTURA DE LA RAM

	if IU_ESTADO=CONTROL then
		if CNT_READ_RAM<(CNT_WRITE_RAM-1) then
			if  UART_ESTADO = STOP then
				nCNT_READ_RAM<=CNT_READ_RAM;---------------------------------
----------------------------------
			else
				nCNT_READ_RAM<=CNT_READ_RAM+1;------------------------------
------a frecuencia de 100 hz(10 ms)
			end if;
		else nCNT_READ_RAM<=(others=>'0');----------------------------------------------------
		end if;
	elsif IU_ESTADO=RESETBC then
		nCNT_READ_RAM<=(others=>'0');
	elsif (IU_ESTADO=ESPERAR or IU_ESTADO=BAJAR_TOPE)  then
		if (CNT_READ_RAM<1) and (CNT_WRITE_RAM>0) then---------------------solo
para sacar el primer dato
			nCNT_READ_RAM<=CNT_READ_RAM+1;------------------------------------a
frecuencia de 100 hz(10 ms)
		else nCNT_READ_RAM<=CNT_READ_RAM;
		end if;
	end if;
end process;

-- Confirmacion de la recepción del dato y guardado en memoria

process (CLK,DRX,IU_ESTADO,EN_CLKRx,we,UART_ESTADO,CNT_WRITE_RAM)
begin
	if (CLK'event and CLK='1') then --Registros
```

```
			if EN_CLKRx='1' then RD<='0';
			elsif (we = '1') then-----------------------------------------------------------------------
			RD<='1';
			end if;
			if (we = '1') and UART_ESTADO=LEERDATOS then
				ram_block(conv_integer(CNT_WRITE_RAM)) <= DRX(4 downto 0);
			end if;
		end if;
end process;
```

-- Unión de los 4 bits más significativos con los menos significativos en un registro para su posterior utilización

```
process (data,CNT_READ_RAM,SINCRONIZADOR_RAM,IU_ESTADO)
begin
		nSINCRONIZADOR_RAM<=SINCRONIZADOR_RAM;
		if CNT_READ_RAM>0 then
			if ((CONV_INTEGER (CNT_READ_RAM)) mod 2) = 1 then
				nSINCRONIZADOR_RAM(9 downto
5)<=(SIGNED(ram_block(CONV_INTEGER(CNT_READ_RAM-1))));
				nSINCRONIZADOR_RAM(4 downto
0)<=(SIGNED(ram_block(CONV_INTEGER(CNT_READ_RAM))));
			end if;
		end if;
end process;
```

-- Elección de consigna dependiendo de IU_ESTADO

```
process
(SALIDA_RAM,IU_ESTADO,CONSIGNAx256,CONSIGNABAJARx256,CNT13,UART_ESTADO)
begin
nCONSIGNABAJARx256<=CONSIGNABAJARx256;
nCONSIGNAx256<=CONSIGNAx256;
nCNT13<=CNT13;
		if IU_ESTADO=BAJAR_TOPE then
			nCNT13<=CNT13+1;
			if UART_ESTADO=STOP then
			nCONSIGNABAJARx256<=CONSIGNABAJARx256;
			else
			nCONSIGNABAJARx256<=CONSIGNABAJARx256-205;
			end if;
			CONSIGNA<=CONSIGNABAJARx256(21 downto 8);
		elsif IU_ESTADO=ESPERAR then
			nCNT13<=CNT13+1;
```

```
				CONSIGNA<=(others=>'0');---POSICION 0
		elsif IU_ESTADO=AJUSTAR then
				if UART_ESTADO=STOP then
				nCONSIGNAx256<=CONSIGNAx256;
				else
				nCONSIGNAx256<=CONSIGNAx256+SALIDA_RAM;
				end if;
				CONSIGNA<=CONSIGNAx256(21 downto 8);
		elsif IU_ESTADO=CONTROL then
				CONSIGNA<=SALIDA_RAM;
		else
				nCONSIGNAx256<=(others=>'0');
				nCONSIGNABAJARx256<="0010011110001000000000";--2530+8ceros=>
"111111111111111"=32767/12=2730-200=2530=>Deshaciendo el bucle
				CONSIGNA<="00100111100010";--2530=>+200=2730*12=32760
				nCNT13<=(others=>'0');
		end if;
end process;

-- Cálculos para la diferencia entre la posicion deseada y el encoder

OPERACION1<=conv_signed(CONSIGNA,14);
OPERACION2<=OPERACION1+200;
OPERACION3<=OPERACION2(10 downto 0)&"0000";
OPERACION4<=OPERACION2(12 downto 0)&"00";
POS_DESEADA<=OPERACION3-OPERACION4;
DIFERENCIA<=(SIGNED('0'& POS_DESEADA)-SIGNED('0'& CNT_ENC(14 downto 0)));

-- multiplicación por la constante del regulador y extensión del signo
--x0,5 para el motor suelto (es decir quitar el último bit, dividir entre 2)
--x0,125 para el motor con toda la estructura (es decir quitar los 3 últimos bits,dividir entre 3)

process (DIFERENCIA)
begin
		if DIFERENCIA>=0 then
				ERROR<="000"&DIFERENCIA(15 downto 3);
		else
				ERROR<="111"&DIFERENCIA(15 downto 3);
		end if;
end process;

-- Acotación del error para evitar desbordamiento

process (ERROR)
```

```
begin
	if (ERROR)>=100 then ERROR_ACOTADO<=conv_signed(100,16);
	elsif (ERROR)<=-100 then ERROR_ACOTADO<=conv_signed(-100,16);
	else ERROR_ACOTADO<=ERROR;
	end if;
end process;

-- Condición para PWM '1' o '0'

process (BUCLE2,CNT_PWM,CNT_REF,ERROR_ACOTADO)
begin
	if BUCLE2='1'	then
		if (CNT_PWM < ABS(ERROR_ACOTADO)) then
			PWM<='1';
		else
			PWM<='0';
		end if;
	elsif BUCLE2='0' then
		if (CNT_PWM < CNT_REF) then
			PWM<='1';
		else
			PWM<='0';
		end if;
	end if;
end process;

-- Máquina de estados del motor según bucle abierto o cerrado

process(IU_ESTADO,UART_ESTADO,FINS_CNT_ENC,FINB_CNT_ENC,CNT_ENC,SW_SUBE2,SW_
BAJA2,SW_STEP2,FUP,FDN,SW_CICLO2,FCS2,FCI2,ILIM2,BUCLE2,ERROR_ACOTADO,CONSIGNA,
SALIDA_RAM,CNT_WRITE_RAM,ram_block,CNT13,CNT_READ_RAM,LD3,LD4,LD5,LD6,LD7)--
>MODIFICADO
begin
  INA<='0';
  INB<='0';
  nIU_ESTADO <= IU_ESTADO;
  nFINS_CNT_ENC <= FINS_CNT_ENC;
  nFINB_CNT_ENC <= FINB_CNT_ENC;
  NLD3<='0';
  NLD4<='0';
  NLD5<='0';
  NLD6<='0';
  NLD7<='0';
```

```
  case IU_ESTADO is

-- Bucle cerrado

        when RESETBC=>
        NLD3<='1';
        NLD4<='0';
        NLD5<='0';
        NLD6<='0';
        NLD7<='0';
         if BUCLE2='0' then nIU_ESTADO<=RESET;
         else
                if  UART_ESTADO=ACABANDATOSRX then-------------------------------------------------
                        if  FCI2='1' then nIU_ESTADO <= BAJAR_TOPE;
                        else nIU_ESTADO <= ESPERAR;
                        end if;
                end if;
         end if;
        when BAJAR_TOPE=>
        NLD3<='0';
        NLD4<='1';
        NLD5<='0';
        NLD6<='0';
        NLD7<='0';
         if ERROR_ACOTADO>0 then
                        INA<='1';
                        INB<='0';
         elsif ERROR_ACOTADO<0 then
                        INA<='0';
                        INB<='1';
         else
                        INA<='0';
                        INB<='0';
         end if;
    if (ILIM2='0') then nIU_ESTADO <= RESET;
         elsif (FCI2='0') then      nIU_ESTADO <= ESPERAR;
         elsif UART_ESTADO=PARAR then nIU_ESTADO<=RESET;
         elsif BUCLE2='0' then nIU_ESTADO<=RESET;
    end if;
        when ESPERAR=>--ESTADO ESPERA PARA QUE HASTA QUE NO SE LLEGUE A 2,56
SEGUNDOS NO SE INICIALICE EL SIGUIENTE ESTADO
        NLD3<='0';
        NLD4<='0';
        NLD5<='1';
```

```
NLD6<='0';
NLD7<='0';
 if ILIM2='0'  then nIU_ESTADO <= RESET;
 elsif CNT13="10100010100" then nIU_ESTADO<=AJUSTAR;--"10100010100" then
nIU_ESTADO<=AJUSTAR;--1300 cada 10 ms=13 segundos
 elsif UART_ESTADO=PARAR then nIU_ESTADO<=RESET;
 elsif BUCLE2='0' then nIU_ESTADO<=RESET;
 end if;
when AJUSTAR=>
NLD3<='0';
NLD4<='0';
NLD5<='0';
NLD6<='1';
NLD7<='0';
 if ERROR_ACOTADO>0 then
        INA<='1';
        INB<='0';
 elsif ERROR_ACOTADO<0 then
        INA<='0';
        INB<='1';
 else
        INA<='0';
        INB<='0';
 end if;
 if ILIM2='0'  then nIU_ESTADO <= RESET;
 elsif CONSIGNA=SALIDA_RAM then nIU_ESTADO<=CONTROL;
 elsif UART_ESTADO=PARAR then nIU_ESTADO<=RESET;
 elsif BUCLE2='0' then nIU_ESTADO<=RESET;
 else nIU_ESTADO<=AJUSTAR;
 end if;
 ---------------------------------------------------escribir algo
when CONTROL=>
NLD3<='0';
NLD4<='0';
NLD5<='0';
NLD6<='0';
NLD7<='1';
 if ERROR_ACOTADO>0 then
        INA<='1';
        INB<='0';
 elsif ERROR_ACOTADO<0 then
        INA<='0';
        INB<='1';
 else
```

```
                        INA<='0';
                        INB<='0';
            end if;
            if ILIM2='0'  then nIU_ESTADO <= RESET;
            elsif BUCLE2='0' then nIU_ESTADO<=RESET;
            elsif UART_ESTADO=PARAR then nIU_ESTADO<=RESET;
            end if;

-- Bucle abierto (sin modificar)

   when RESET =>
            if BUCLE2='1' then nIU_ESTADO<=RESETBC;
            else
                  nFINS_CNT_ENC <= "0111111111111111";
                  nFINB_CNT_ENC <= (others=>'0');
                  if (SW_SUBE2='1' and FCS2='1') then nIU_ESTADO <= SUBIDA;
                  elsif (SW_BAJA2='1' and FCI2='1') then nIU_ESTADO <= BAJADA;
                  elsif (SW_STEP2='1' and FUP='1' and FCS2='1') then
                        nIU_ESTADO <= SUBIDA;
                        nFINS_CNT_ENC <= CNT_ENC + 1200;  --Subir 10 mm desde posicion
actual
                  elsif (SW_STEP2='1' and FDN='1' and FCI2='1') then
                        nIU_ESTADO <= BAJADA;
                        nFINB_CNT_ENC <= CNT_ENC - 1200;  --Subir 10 mm desde posicion
actual
                  elsif (SW_CICLO2='1' and FCS2='1') then nIU_ESTADO <= INI_CICLO;
                  end if;
             end if;
   when INI_CICLO =>
     nFINS_CNT_ENC <= CNT_ENC + 2400;           --Subir 20 mm desde posicion actual
     nFINB_CNT_ENC <= CNT_ENC;          --Luego bajar hasta posicion actual
     nIU_ESTADO <= PARO_I;
   when PARO_I =>
     if (SW_CICLO2='1') then nIU_ESTADO <= SUBIDA;
     else  nIU_ESTADO <= RESET;
     end if;
   when SUBIDA =>
            INA<='1';
     if (ILIM2='0' or FCS2='0') then nIU_ESTADO <= RESET;
     elsif (SW_SUBE2='0' and SW_STEP2='0' and SW_CICLO2='0') then  nIU_ESTADO <= RESET;
     elsif (SW_STEP2='1' and CNT_ENC=FINS_CNT_ENC) then nIU_ESTADO <= RESET;
     elsif (SW_CICLO2='1' and CNT_ENC=FINS_CNT_ENC) then nIU_ESTADO <= PARO_S;
     end if;
   when PARO_S =>
```

```
    if (SW_CICLO2='1') then nIU_ESTADO <= BAJADA;
    else  nIU_ESTADO <= RESET;
    end if;
   when BAJADA =>
          INB<='1';
    if (ILIM2='0' or FCI2='0') then nIU_ESTADO <= RESET;
    elsif (SW_BAJA2='0' and SW_STEP2='0' and SW_CICLO2='0') then  nIU_ESTADO <= RESET;
    elsif (SW_STEP2='1' and CNT_ENC=FINB_CNT_ENC) then nIU_ESTADO <= RESET;
    elsif ((SW_CICLO2='1') and (CNT_ENC=FINB_CNT_ENC)) then nIU_ESTADO <= PARO_I;
    end if;
   when others =>
    nIU_ESTADO <= RESET;
  end case;
end process;

end Behavioral;
```

8.3.3 UART_TX

```
-- Bits datos parametrizable, 1 bit STOP, paridad impar
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity UART_TX is
  generic (DWL1: integer range 0 to 7 := 7);  -- DWLS (Data Width Less 1)
  port (CLK: in std_logic;
        EN   : in std_logic;
        RST  : in std_logic;
        WR   : in std_logic;
        DIN  : in std_logic_vector(DWL1 downto 0);
        Tx   : out std_logic;
        TxRDY: out std_logic);
end;

architecture behavioral of UART_TX is
-- Máquina de estados
  type TIPO_ESTADO is (INICIO, DATOS, PARIDAD, STOP);
  signal ESTADO, nESTADO: TIPO_ESTADO;
-- Registro de desplazamiento
  signal PS : std_logic_vector(DWL1 downto 0);
-- Bit de paridad generado
  signal BIT_PARIDAD : std_logic;
-- Contador bit de dato
```

```
  signal CNTBIT : std_logic_vector(2 downto 0);

begin

-- Proceso secuencial para ESTADO
  process (CLK, RST)
  begin
    if (RST='1') then
      ESTADO <= STOP;
    elsif (CLK'event and CLK='1') then
      if EN='1' then
        ESTADO <= nESTADO;
      end if;
    end if;
  end process;

-- Proceso combinacional para ESTADO
  process (ESTADO, WR, CNTBIT)
  begin
    nESTADO <= ESTADO;
    case ESTADO is
      when INICIO  =>
          nESTADO <= DATOS;
      when DATOS =>
            if CNTBIT=DWL1 then --7
              nESTADO <= PARIDAD;
            end if;
      when PARIDAD =>
            nESTADO <= STOP;
      when STOP =>
            if WR = '1' then
              nESTADO <= INICIO;
            end if;
    end case;
  end process;

-- Contador que determina el bit de DATOS
  process (CLK, RST)
  begin
    if RST='1' then
      CNTBIT <= (others => '0');
    elsif (CLK'event and CLK='1') then
      if EN='1' then
        if ESTADO=DATOS then
```

```
        CNTBIT <= CNTBIT + 1;
      else
        CNTBIT <= (others => '0');
      end if;
    end if;
  end if;
end process;

-- Registro de desplazamiento
process (CLK, RST)
begin
  if RST='1' then
    PS <= (others => '0');
  elsif (CLK'event and CLK='1') then
    if EN='1' then
      if (ESTADO=STOP and WR = '1') then
        PS <= DIN;
      elsif ESTADO=DATOS then
        PS <= '0' & PS(DWL1 downto 1);
      end if;
    end if;
  end if;
end process;

-- Generación de paridad
process (CLK, RST)
begin
  if RST='1' then
    BIT_PARIDAD <= '0';
  elsif (CLK'event and CLK='1') then
    if EN='1' then
      if (ESTADO=DATOS) then
            BIT_PARIDAD <= BIT_PARIDAD xor PS(0);
      else
            BIT_PARIDAD <= '0';
      end if;
    end if;
  end if;
end process;

-- Salidas
Tx <= '0' when ESTADO=INICIO else
      PS(0) when ESTADO=DATOS else
      BIT_PARIDAD when ESTADO=PARIDAD else
```

```
      '1';

  TxRDY <= '1' when ESTADO=STOP else '0';

end behavioral;
```

8.3.4 UART_RX

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity UART_RX is
  port (CLK  : in std_logic;
        EN   : in std_logic;
        RST  : in std_logic;
        Rx   : in std_logic;
        RD   : in std_logic;
        DOUT : out std_logic_vector(7 downto 0);
        ERROR_PARIDAD  : out std_logic;
        ERROR_FORMATO  : out std_logic;
        SOBRESCRITURA  : out std_logic;
        RxRDY: out std_logic);
end;

architecture behavioral of UART_RX is
-- Máquina de estados
  type TIPO_ESTADO is (INICIO,DATOS,PARIDAD,STOP);
  signal ESTADO, ESTADO_SIG: TIPO_ESTADO;
-- Contador de periodos de reloj
  signal CNTCLK     :std_logic_vector(3 downto 0);
-- Contador del bit de dato
  signal CNTBIT     :std_logic_vector(2 downto 0);
-- Sincronizadores
  signal Rx_REG, Rx_REG2    : std_logic;
-- Confirmación de dato recibido
  signal RxRDY_SIG, RxRDY_REG  : std_logic;
-- Registro de desplazamiento serie/paralelo
  signal REG_DESP : std_logic_vector(7 downto 0);
-- Registro de salida
  signal DOUT_SIG, DOUT_REG : std_logic_vector(7 downto 0);
-- Bit de paridad generado
  signal BIT_PARIDAD : std_logic;
```

```
-- Señal de igual paridad
  signal IGUAL_PARIDAD : std_logic;

begin

-- Sincronizador para evitar metaestabilidad
  process (CLK, RST)
  begin
    if (RST='1') then
      Rx_REG  <= '1';
      Rx_REG2 <= '1';
    elsif (CLK'event and CLK='1') then
      Rx_REG2 <= Rx_REG;
      Rx_REG  <= Rx;
    end if;
  end process;

-- Proceso secuencial para ESTADO
  process (CLK, RST)
  begin
    if (RST='1') then
      ESTADO   <= STOP;
    elsif (CLK'event and CLK='1') then
      if EN='1' then
        ESTADO <= ESTADO_SIG;
      end if;
    end if;
  end process;

-- Proceso combinacional para ESTADO
  process (ESTADO, CNTBIT, Rx_REG2, CNTCLK)
  begin
    ESTADO_SIG <= ESTADO;  -- valor por defecto
    case ESTADO is
      when INICIO  =>
              if CNTCLK=7 and Rx_REG2 = '1' then
                ESTADO_SIG <= STOP;
              elsif CNTCLK=15 then
                ESTADO_SIG <= DATOS;
              end if;
      when DATOS =>
              if CNTCLK=15 and CNTBIT=7 then
                ESTADO_SIG <= PARIDAD;
              end if;
```

```
    when PARIDAD =>
            if CNTCLK=15 then
              ESTADO_SIG <= STOP;
            end if;
    when STOP =>
            if Rx_REG2 = '0' then
              ESTADO_SIG <= INICIO;
            end if;
  end case;
 end process;

-- Contador de reloj
 process (CLK, RST)
 begin
  if (RST='1') then
    CNTCLK <= (others => '0');
  elsif (CLK'event and CLK='1') then
    if EN='1' then
      if ESTADO=STOP and Rx_REG2 = '1' then
              CNTCLK <= (others => '0');
      else
              CNTCLK <= CNTCLK+1;
      end if;
    end if;
  end if;
 end process;

-- Contador del bit de dato
 process (CLK, RST)
 begin
  if (RST='1') then
    CNTBIT <= (others => '0');
  elsif (CLK'event and CLK='1') then
    if EN='1' and CNTCLK=15 then
      if (ESTADO/=DATOS) then
              CNTBIT <= (others => '0');
      else
        CNTBIT <= CNTBIT+1;
      end if;
    end if;
  end if;
 end process;

-- Determina el bit de paridad
```

```
  process (CLK, RST)
  begin
    if RST='1' then
      BIT_PARIDAD  <= '0';
    elsif (CLK'event and CLK='1') then
            if ESTADO=STOP then --MODIFICADO
        BIT_PARIDAD <= '0';
      elsif EN='1' and CNTCLK=7 then
                  if ESTADO=PARIDAD then
        BIT_PARIDAD <= BIT_PARIDAD;
                  else
        BIT_PARIDAD <= BIT_PARIDAD xor Rx_REG2;
       end if;
      end if;
    end if;
  end process;

-- Señal de error de paridad
  IGUAL_PARIDAD <= '1' when (ESTADO=PARIDAD and BIT_PARIDAD=Rx_REG2) else '0';

-- Rellena el registro de desplazamiento
  process (CLK, RST)
  begin
    if RST='1' then
      REG_DESP <= (others => '0');
    elsif (CLK'event and CLK='1') then
      if EN='1' and ESTADO=DATOS and CNTCLK=7 then
        REG_DESP <= Rx_REG2 &  REG_DESP(7 downto 1);
      end if;
    end if;
  end process;

-- Determina las salidas DOUT y RxRDY
  process (CLK, RST)
  begin
    if RST='1' then
      DOUT_REG <= (others => '0');
      RxRDY_REG  <= '0';
    elsif (CLK'event and CLK='1') then
      if EN='1' then
        DOUT_REG <= DOUT_SIG;
        RxRDY_REG <= RxRDY_SIG;
      end if;
    end if;
```

```
  end process;

-- Determina las salidas DOUT y RxRDY
  process (CNTCLK, ESTADO, REG_DESP, IGUAL_PARIDAD, DOUT_REG)
  begin
    if ESTADO=PARIDAD and CNTCLK=7 then
      if IGUAL_PARIDAD = '1' then
        DOUT_SIG <= REG_DESP;
      else
                    DOUT_SIG <= (others => '0');
      end if;
    else
      DOUT_SIG <= DOUT_REG;
    end if;
  end process;

-- Determina las salidas DOUT y RxRDY
  process (RD, CNTCLK, ESTADO, IGUAL_PARIDAD, RxRDY_REG)
  begin
    if RD='1' then
       RxRDY_SIG <= '0';
    elsif ESTADO=PARIDAD and CNTCLK=7 then
      if IGUAL_PARIDAD = '1' then
        RxRDY_SIG <= '1';
      else
        RxRDY_SIG <= '0';
      end if;
    else
      RxRDY_SIG <= RxRDY_REG;
    end if;
  end process;

-- Salidas
  RxRDY <= RxRDY_REG;
  DOUT  <= DOUT_REG;
  SOBRESCRITURA <= '1' when (ESTADO=PARIDAD and CNTCLK=7 and IGUAL_PARIDAD = '1' and
RxRDY_REG='1') else '0';
  ERROR_PARIDAD <= '1' when (ESTADO=PARIDAD and CNTCLK=7 and
BIT_PARIDAD/=Rx_REG2) else '0';
  ERROR_FORMATO <= '1' when (ESTADO=STOP and CNTCLK=7 and Rx_REG2='0') else '0';

end behavioral;
```

8.4 Programación MATLAB

8.4.1 SENOIDE

```
clc
clear all
close all

PS=serial('COM4');
set(PS,'Baudrate',57600); % se configura la velocidad a 57600 Baudios
set(PS,'StopBits',1); % se configura bit de parada a uno
set(PS,'DataBits',8); % se configura que el dato es de 8 bits
set(PS,'Parity','even'); % se configura sin paridad
set(PS,'FlowControl','none');%Sin control de hardware

estado=8;
while estado~=6
        if estado==8
        a=input('ingrese el valor de amplitud de la senoide = '); %amplitud
        d=input('ingrese el número de divisiones = '); %divisiones
        puntoinicio=input('ingrese el punto de inicio (de 0 a 1000)= '); %puntoinicio
        x=linspace(0,2*pi,d);
        y=a*sin(x)+puntoinicio;
                d=d-1;
                for i=1:d
                j=round(y(i));
                q(i)=j;
                binario=fi(j,0,10,0);
                if i==1
                        partebajabyte=bitget(binario,[5:-1:1]);
                        %disp(bin(partebajabyte))

                        partealtabyte=bitget(binario,[10:-1:6]);
                        %disp(bin(partealtabyte))

                        enviar=32+bin2dec(bin(partealtabyte));%+32=> añadimos '001'
como bits más significativos

                        fopen(PS); %Abre objeto
                        fwrite(PS,enviar);

                        enviar=bin2dec(bin(partebajabyte));

                        fwrite(PS,enviar);
```

```
                    end
                    if i>2
                    partebajabyte=bitget(binario,[5:-1:1]);
                    %disp(bin(partebajabyte))

                    partealtabyte=bitget(binario,[10:-1:6]);
                    %disp(bin(partealtabyte))

                    enviar=bin2dec(bin(partealtabyte));

                    fwrite(PS,enviar);

                    enviar=bin2dec(bin(partebajabyte));

                    fwrite(PS,enviar);

                    end
                    end
                    fid=fopen('archivoconsigna.txt','w');
                    fprintf(fid,'%i \n',q);

                    load archivoconsigna.txt
                    figure
                    plot(archivoconsigna)
                    fclose(fid);

                    enviar=64;

                    fwrite(PS,enviar);
                    fclose(PS);


elseif estado==1
                    enviar=128;

                    fopen(PS); %Abre objeto
                    fwrite(PS,enviar);
                    fclose(PS);

elseif estado==2
                    enviar=96;

                    fopen(PS); %Abre objeto
                    fwrite(PS,enviar);
```

```
                    enviar=0;
                    fwrite(PS,enviar);
                    fclose(PS);

elseif estado==3
                    enviar=160;


                    fopen(PS); %Abre objeto
                    fwrite(PS,enviar);
                    fclose(PS);
elseif estado==4
                    enviar=192;

                    fopen(PS); %Abre objeto
                    fwrite(PS,enviar);
                    fclose(PS);
elseif estado==7
                    fopen(PS);
                    sizebuffer=6;
                    l=0;
                    n=0;
                    datosaleer=input('ingrese el número de datos que quiere leer = ');
                    intnumerocompleto=2;
                    while intnumerocompleto~=1
                              datos= fread(PS,sizebuffer,'uchar');
                              binary=fi(datos,0,8,0);
                              for h=1:sizebuffer
                                        if mod(h,6)==3
                                                  if mod(h,2)==1

numerocompleto=bitconcat(binary(h),binary(h+1));
                                                  %disp(bin(numerocompleto))

intnumerocompleto=bin2dec(bin(numerocompleto));
                                                  end
                                        end
                              end
                    end
                    for l=1:datosaleer
                              for h=1:sizebuffer
                                        if mod(h,6)==1
                                                  if mod(h,2)==1
```

```
          numerocompleto=bitconcat(binary(h),binary(h+1));
                                             %disp(bin(numerocompleto))

          intnumerocompleto=bin2dec(bin(numerocompleto));%--------------
                                             posreal=intnumerocompleto/12-200;
                                             z(l)=posreal;
                                             end
                                   end
                         end
                    datos= fread(PS,sizebuffer,'uchar');
                    binary=fi(datos,0,8,0);
                    end
                    fid=fopen('archivoencoder.txt','w');
                    fprintf(fid,'%i \n',z);

                    load archivoencoder.txt
                    figure
                    plot(archivoencoder)
                    fclose(fid);
                    clear z;
                    fclose(PS);
          end

estado=input('INTRODUZCA EL NÚMERO.- 1 stop,2 start,3 mandar_posicion_on,4
mandar_posicion_off,5 volver_a_mandar,6 salir,7 ver posición = ');
          if estado==5
                    enviar=224;%ASI TERMINA EL ESTADO EN EL QUE ESTE Y EMPIEZA EN
RESET

                    fopen(PS); %Abre objeto
                    fwrite(PS,enviar);
                    fclose(PS);
                    estado=8;
          end
end


%fclose(PS);
delete(PS);
clear PS
fprintf('ok');
```

```
%enviar='001'=32
%acabar='010'=64
%stop='100'=128
%start'011'=96
%madarposon='101'=160
%mandarposoff='110'=192
%parar='111'=224
```

8.4.2 ESCALÓN

```
clc
clear all
close all

PS=serial('COM4');
set(PS,'Baudrate',57600); % se configura la velocidad a 57600 Baudios
set(PS,'StopBits',1); % se configura bit de parada a uno
set(PS,'DataBits',8); % se configura que el dato es de 8 bits, debe estar entre 5 y 8
set(PS,'Parity','even'); % se configura sin paridad
set(PS,'FlowControl','none');%Sin control de hardware, o software to

estado=8;
while estado~=6
        if estado==8
        al=input('ingrese el valor de la altura del escalón = '); %altura
        d=input('ingrese el número de divisiones = '); %divisiones
        puntoinicio=input('ingrese el punto de inicio = '); %punto de inicio
        referencia=puntoinicio;
        p1=round(d/10);
        p2=round(6*d/10);
        for i=1:d
        axis([0 d puntoinicio-al-1 puntoinicio+al+1]);
        t(i)=puntoinicio;
                if i==p1
                puntoinicio=puntoinicio+al;
                elseif i==p2
                puntoinicio=puntoinicio-al;
                end
                        binario=fi(puntoinicio,0,10,0);
                        if i==1
                                partebajabyte=bitget(binario,[5:-1:1]);
                                %disp(bin(partebajabyte))

                                partealtabyte=bitget(binario,[10:-1:6]);
```

```
				%disp(bin(partealtabyte))

				enviar=32+bin2dec(bin(partealtabyte));%+32=> añadimos '001'
como bits más significativos

				fopen(PS); %Abre objeto
				fwrite(PS,enviar);

				enviar=bin2dec(bin(partebajabyte));

				fwrite(PS,enviar);

			end
			if i>1
			partebajabyte=bitget(binario,[5:-1:1]);
			%disp(bin(partebajabyte))

			partealtabyte=bitget(binario,[10:-1:6]);
			%disp(bin(partealtabyte))

			enviar=bin2dec(bin(partealtabyte));

			fwrite(PS,enviar);

			enviar=bin2dec(bin(partebajabyte));

			fwrite(PS,enviar);

			end
	end
	fid=fopen('archivoconsigna.txt','w');
	fprintf(fid,'%i \n',t);
	load archivoconsigna.txt
	figure
	plot(archivoconsigna)
	fclose(fid);

			enviar=64;

			fwrite(PS,enviar);
			fclose(PS);


	elseif estado==1
```

```
            enviar=128;

            fopen(PS); %Abre objeto
            fwrite(PS,enviar);
            fclose(PS);

elseif estado==2
            enviar=96;

            fopen(PS); %Abre objeto
            fwrite(PS,enviar);
            enviar=0;
            fwrite(PS,enviar);
            fclose(PS);

elseif estado==3
            enviar=160;


            fopen(PS); %Abre objeto
            fwrite(PS,enviar);
            fclose(PS);
elseif estado==4
            enviar=192;

            fopen(PS); %Abre objeto
            fwrite(PS,enviar);
            fclose(PS);
elseif estado==7
            fopen(PS);
            sizebuffer=6;
            l=0;
            n=0;
            datosaleer=input('ingrese el número de datos que quiere leer = ');
            intnumerocompleto=2;
            while intnumerocompleto~=1
                  datos= fread(PS,sizebuffer,'uchar');
                  binary=fi(datos,0,8,0);
                  for h=1:sizebuffer
                        if mod(h,6)==3
                              if mod(h,2)==1

numerocompleto=bitconcat(binary(h),binary(h+1));
                              %disp(bin(numerocompleto))
```

```
        intnumerocompleto=bin2dec(bin(numerocompleto));%-----------------
                                        end
                                    end
                                end
                            end
                            for l=1:datosaleer
                                for h=1:sizebuffer
                                    if mod(h,6)==1
                                        if mod(h,2)==1

        numerocompleto=bitconcat(binary(h),binary(h+1));
                                        %disp(bin(numerocompleto))

        intnumerocompleto=bin2dec(bin(numerocompleto));
                                        posreal=intnumerocompleto/12-200;
                                        z(l)=posreal;
                                        end
                                    end
                                end
                            datos= fread(PS,sizebuffer,'uchar');
                            binary=fi(datos,0,8,0);
                            end
                            fid=fopen('archivoencoder.txt','w');
                            fprintf(fid,'%i \n',z);

                            load archivoencoder.txt
                            figure
                            plot(archivoencoder)
                            fclose(fid);
                            clear z;
                            fclose(PS);
        end
estado=input('INTRODUZCA EL NÚMERO.- 1 stop,2 start,3 mandar_posicion_on,4
mandar_posicion_off,5 volver_a_mandar,6 salir,7 ver posición = ');
        if estado==5
                            enviar=224;%ASI TERMINA EL ESTADO EN EL QUE ESTE Y EMPIEZA EN
RESET

                            fopen(PS); %Abre objeto
                            fwrite(PS,enviar);
                            fclose(PS);
                            estado=8;
        end
```

```
end


%fclose(PS);
delete(PS);
clear PS;
fprintf('ok');


%enviar='001'=32
%acabar='010'=64
%stop='100'=128
%start'011'=96
%madarposon='101'=160
%mandarposoff='110'=192
```

8.5 Bibliografía

ELECTRÓNICA DIGITAL APLICACIONES Y PROBLEMAS / José Ignacio Artigas, Luis Ángel Barragán, Carlos Orrite, Isidro Urriza.

Apuntes de Microelectrónica.

Manual de ayuda de MATLAB.

Memoria proyecto fin de carrera control con FPGA de actuador lineal para emulador de equinoterapia / Javier Marco Estruc.

8.6 Programas utilizados.

ISE 14.3

Matlab 2010

Realterm

Hyperterminal

Scilab 5.4.0

Adept

Eagle 6.1.0